



**UNIVERSITAS INDONESIA**

**PERANCANGAN DAN IMPLEMENTASI *BORDER ROUTER*  
SEBAGAI HTTP-COAP *PROXY* UNTUK JARINGAN SENSOR  
DAN AKTUATOR NIRKABEL**

**SKRIPSI**

**ADIKA BINTANG SULAEMAN  
1206243053**

**FAKULTAS TEKNIK  
PROGRAM STUDI TEKNIK KOMPUTER  
DEPOK  
JUNI 2016**



**UNIVERSITAS INDONESIA**

**PERANCANGAN DAN IMPLEMENTASI *BORDER ROUTER*  
SEBAGAI HTTP-COAP *PROXY* UNTUK JARINGAN SENSOR  
DAN AKTUATOR NIRKABEL**

**SKRIPSI**

**Diajukan sebagai salah satu syarat untuk memperoleh gelar Sarjana Teknik**

**ADIKA BINTANG SULAEMAN  
1206243053**

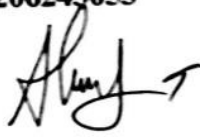
**FAKULTAS TEKNIK  
PROGRAM STUDI TEKNIK KOMPUTER  
DEPOK  
JUNI 2016**

## **HALAMAN PERNYATAAN ORISINALITAS**

**Skripsi ini adalah hasil karya saya sendiri,  
dan semua sumber baik yang dikutip maupun dirujuk  
telah saya nyatakan dengan benar.**

**Nama : Adika Bintang Sulaeman**

**NPM : 1206243053**

**Tanda Tangan : **

**Tanggal : 15 Juli 2016**

## HALAMAN PENGESAHAN

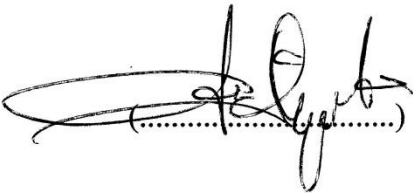
Skripsi ini diajukan oleh:

Nama : Adika Bintang Sulaeman  
NPM : 1206243053  
Program Studi : Teknik Komputer  
Judul Skripsi : Perancangan dan Implementasi *Border Router*  
Sebagai HTTP-CoAP *Proxy* Untuk Jaringan  
Sensor dan Aktuator Nirkabel


Telah berhasil dipertahankan di hadapan Dewan Penguji dan diterima sebagai bagian persyaratan yang diperlukan untuk memperoleh gelar Sarjana Teknik pada Program Studi Teknik Komputer, Fakultas Teknik, Universitas Indonesia.

## DEWAN PENGUJI

Pembimbing :  
F. Astha Ekadiyanto, S.T., M.Sc.

  
(.....)

Penguji :  
Prof. Dr.-Ing. Ir. Kalamullah Ramli, M.Eng.

  
(.....)

Penguji :  
Prof. Dr. Ir. Riri Fitri Sari, M.Sc., M.M.

  
(.....)

Ditetapkan di : Depok

Tanggal : 15 Juli 2016

## **KATA PENGANTAR**

Puji syukur penulis panjatkan kepada Allah SWT sebab atas segala rahmat dan karunia-Nya, penulis dapat menyelesaikan seminar ini dengan baik. Penulis menyadari bahwa seminar ini tidak dapat diselesaikan tanpa bantuan dari banyak pihak. Oleh karena itu, penulis mengucapkan terima kasih kepada:

1. Bapak F. Astha Ekadiyanto, S.T., M.Sc. selaku pembimbing seminar yang telah memberikan pengarahan, koreksi, dukungan, dan waktu selama penulis mengerjakan seminar ini.
2. Orang tua dan keluarga penulis yang telah memberikan dukungan baik moril maupun materil sehingga penulis dapat menyelesaikan seminar ini.
3. Pemberi dukungan moral dan motivasi, Melliawati.
4. Teman seperjuangan skripsi dibawah bimbingan F. Astha Ekadiyanto, S.T., M.Sc., yaitu Rafi Kurnia Putra dan Yudi Andrean.
5. Teman diskusi yang sangat membantu dalam proses pengerjaan seminar dan pembelian komponen elektronik, Aldwin Akbar.
6. Teman-teman Tim Robotika UI dan tim lomba MEMCE2 yaitu Sanadhi, Jendra, Farid, Harianto, dan yang lainnya yang tidak dapat disebut satu per satu.

Akhir kata, semoga Tuhan berkenan membalas kebaikan dari semua pihak yang telah berbaik hati membantu penulis dan semoga seminar ini dapat bermanfaat bagi pengembangan teknologi dan ilmu pengetahuan.

Depok, 1 Juli 2016

Adika Bintang Sulaeman

**HALAMAN PERNYATAAN PERSETUJUAN PUBLIKASI TUGAS  
AKHIR UNTUK KEPENTINGAN AKADEMIS**

---

Sebagai sivitas akademik Universitas Indonesia, saya yang bertanda tangan di bawah ini:

Nama : Adika Bintang Sulaeman  
NPM : 1206243053  
Program Studi : Teknik Komputer  
Departemen : Teknik Elektro  
Fakultas : Teknik  
Jenis Karya : Skripsi

demikian pengembangan ilmu pengetahuan, menyetujui untuk memberikan kepada Universitas Indonesia **Hak Bebas Royalti Noneksklusif** (*Non-exclusive Royalty-Free Right*) atas karya ilmiah saya yang berjudul:

**“Perancangan dan Implementasi *Border Router* Sebagai HTTP-CoAP Proxy  
Untuk Jaringan Sensor dan Aktuator Nirkabel”**

Dengan Hak Bebas Royalti Noneksklusif ini Universitas Indonesia berhak menyimpan, mengalih media/formatkan, mengelola dalam bentuk pangkalan data (*database*), merawat, dan mempublikasikan tugas akhir saya selama tetap mencantumkan nama saya sebagai penulis/pencipta dan sebagai pemilik Hak Cipta.

Demikian pernyataan ini saya buat dengan sebenar-benarnya.

Dibuat di: Depok

Pada tanggal: 15 Juli 2016

Yang menyatakan



(Adika Bintang Sulaeman)

## ABSTRAK

Nama : Adika Bintang Sulaeman  
Program Studi : Teknik Komputer  
Judul : Perancangan dan Implementasi *Border Router* Sebagai HTTP-CoAP *Proxy* Untuk Jaringan Sensor dan Aktuator Nirkabel

Salah satu tantangan dalam mengintegrasikan jaringan sensor nirkabel dengan internet adalah dibutuhkan mekanisme perantara dua protokol yang berbeda, yaitu HTTP dan CoAP, agar sensor dapat diakses menggunakan URL standar. Mekanisme tersebut dapat diimplementasikan menggunakan *cross-protocol reverse proxy* yang ditanam di dalam *border router*. Penelitian skripsi ini fokus pada perancangan dan penerapan *border router* dengan membangun prototipe dengan fitur pemetaan HTTP-CoAP dan mekanisme *caching*. Hasil dari penelitian ini menunjukkan bahwa *proxy* yang didesain dapat menangani 23 permintaan per detik, memiliki karakter meningkatnya nilai *latency* dengan konstanta kemiringan 23.667 terhadap peningkatan jumlah *client*, memiliki *cache* yang dapat mengurangi *latency*, dan mulai memberi *error* jika diakses lebih dari 1000 *clients* secara bersamaan.

Kata kunci: *Internet of Things, HTTP, CoAP, 6LoWPAN, Cross-Proxy*

## ABSTRACT

Name : Adika Bintang Sulaeman  
Major : Computer Engineering  
Title : Design and Implementation of Border Router as HTTP-CoAP Proxy for Wireless Sensor and Actuator Networks

One of the main challenges in integrating wireless sensor networks with the internet is the need of intermediary mechanism interconnecting two different protocols, i.e. HTTP and CoAP, so that users can access the sensors with the standard URL. Such mechanism can be implemented using cross-protocol reverse proxy which lies on the border router. This research focused on the design and implementation of the border router by building the prototype for wireless sensor networks with HTTP-CoAP mapping and caching mechanism. The result of this research showed that the designed proxy was able to handle 23 requests per second. The proxy had the linear increment of latency in respect to the number of clients with the gradient value of 23.667. The caching mechanism effectively reduced the latency and the proxy started to fail if accessed by more than 1000 clients.

Keywords: *Internet of Things, HTTP, CoAP, 6LoWPAN, Cross-Proxy*



## DAFTAR ISI

<b>HALAMAN PERNYATAAN ORISINALITAS .....</b>	<b>i</b>
<b>HALAMAN PENGESAHAN .....</b>	<b>ii</b>
<b>KATA PENGANTAR.....</b>	<b>iii</b>
<b>HALAMAN PERNYATAAN PERSETUJUAN PUBLIKASI TUGAS AKHIR UNTUK KEPENTINGAN AKADEMIS .....</b>	<b>iv</b>
<b>ABSTRAK .....</b>	<b>v</b>
<b>ABSTRACT .....</b>	<b>vi</b>
<b>DAFTAR ISI.....</b>	<b>vii</b>
<b>DAFTAR GAMBAR.....</b>	<b>x</b>
<b>DAFTAR TABEL .....</b>	<b>xii</b>
<b>BAB 1 PENDAHULUAN .....</b>	<b>1</b>
1.1    Latar Belakang .....	1
1.2    Tujuan Penelitian.....	2
1.3    Batasan Masalah.....	2
1.4    Metode Penelitian.....	3
1.5    Sistematika Penulisan.....	3
<b>BAB 2 TINJAUAN PUSTAKA.....</b>	<b>4</b>
2.1    Protokol Komunikasi IEEE 802.15.4 .....	4
2.1.1    Pengelompokan Peralatan Dalam Jaringan .....	4
2.1.2    Topologi Jaringan IEEE 802.15.4.....	5
2.1.3    Lapisan Fisik IEEE 802.15.4 .....	7
2.1.4 <i>Medium Access Control Layer (MAC Layer)</i> IEEE 802.15.4 .....	8
2.2 <i>IPv6 over Low Power Wireless Personal Area Networks (6LoWPAN)</i> ..	9
2.2.1    Arsitektur 6LoWPAN .....	12
2.2.2    Integrasi Jaringan IP dengan 6LoWPAN dengan <i>Border Router</i> ...	13
2.2.3    Mengimplementasikan 6LoWPAN .....	16
2.3 <i>Constrained Application Protocol (CoAP)</i> .....	18

2.3.1	<i>Cross-Protocol Proxy</i> .....	20
2.3.2	<i>URI Mapping</i> .....	21
2.3.3	<i>Caching</i> .....	22
2.4	<i>Socket Programming</i> .....	23
2.5	<i>Threading</i> .....	24
2.6	Perangkat Keras dan Lunak yang Digunakan .....	25
2.6.1	Raspberry Pi Model B rev 2.....	25
2.6.2	<i>Wi-Fi Dongle USB Adapter</i> .....	26
2.6.3	Modul Radio MRF24J40MA .....	26
2.6.4	Libcoap.....	29
2.6.5	Apache2 Web Server, <i>URL Rewriting</i> , dan CGI.....	30
<b>BAB 3 PERANCANGAN HTTP-COAP PROXY .....</b>		<b>31</b>
3.1	Tinjauan Perancangan <i>Border Router</i> yang Sudah Ada.....	31
3.2	<i>Requirement Analysis</i> (Pertimbangan Perancangan).....	32
3.2.1	Skenario Fungsionalitas Sistem .....	32
3.2.2	Fitur/ <i>Resource</i> yang dimiliki CoAP Server .....	36
3.3	Perancangan Sistem dan Perangkat Lunak.....	36
3.4	Perancangan Perangkat Keras .....	44
3.5	Skenario Pengujian dan Pengukuran Performa HTTP-CoAP Proxy .....	46
<b>BAB 4 IMPLEMENTASI DAN EVALUASI .....</b>		<b>48</b>
4.1.	Implementasi Sistem .....	48
4.1.1	Konfigurasi Raspberry Pi untuk 6LoWPAN <i>Interface</i> .....	48
4.1.2	Implementasi Perangkat Keras.....	50
4.1.3	Implementasi Perangkat Lunak.....	54
4.2	Tes Fungsionalitas .....	60
4.3	Evaluasi Performa .....	63

4.3.1	Evaluasi <i>Latency</i> untuk <i>Concurrent Client</i> dengan <i>Load Testing</i> ..	64
4.3.2	Evaluasi <i>Throughput</i> Dengan <i>Load Testing</i> .....	65
4.3.3	Evaluasi Penggunaan <i>Cache</i> .....	66
4.3.4	Evaluasi <i>Error Rate</i> yang Dihasilkan dengan <i>Stress Testing</i> .....	68
<b>BAB 5 KESIMPULAN DAN SARAN .....</b>		<b>71</b>
<b>DAFTAR REFERENSI .....</b>		<b>73</b>
<b>LAMPIRAN.....</b>		<b>77</b>
LAMPIRAN 1 .....		77

## DAFTAR GAMBAR

Gambar 2.1 Topologi Star [7] .....	6
Gambar 2.2 Topologi Mesh [7].....	6
Gambar 2.3 Jumlah <i>Channel</i> Untuk Tiap Frekuensi [7] .....	7
Gambar 2.4 Mode Operasi IEEE 802.15.4 [7].....	9
Gambar 2.5 IP dan 6LoWPAN Protocol Stacks [3].....	10
Gambar 2.6 Arsitektur 6LoWPAN [3].....	13
Gambar 2.7 <i>Border Router</i> Dengan Antarmuka 6LoWPAN [3] .....	14
Gambar 2.8 Model <i>System-on-a-chip</i> [3].....	16
Gambar 2.9 Model <i>dual-chip</i> [3] .....	17
Gambar 2.10 Model <i>Network Processor</i> [3] .....	17
Gambar 2.11 Constrained RESTful Environment architecture [12].....	20
Gambar 2.12 Skenario HTTP-CoAP <i>reverse proxy</i> [14] .....	21
Gambar 2.13 Modul Microchip MRF24J40MA [22] .....	27
Gambar 2.14 Petunjuk <i>routing</i> modul MRF24J40MA [23] .....	28
Gambar 2.15 Gambar pola radiasi dalam bentuk 3D [23] .....	29
 Gambar 3.1 Skenario <i>Deployment</i> Perancangan.....	 33
Gambar 3.2 Diagram <i>Use Case</i> dari Proxy .....	34
Gambar 3.3 Mekanisme URL Untuk CoAP Server .....	34
Gambar 3.4 Respon CoAP <i>Resource Discovery</i> .....	35
Gambar 3.5 Bentuk JSON dari CoAP <i>Resource Discovery</i> .....	35
Gambar 3.6 <i>Protocol Stack</i> dari Sistem .....	36
Gambar 3.7 Diagram <i>Sequence</i> dari Proxy .....	37
Gambar 3.8 Diagram Blok Sistem <i>Proxy</i> .....	38
Gambar 3.9 Flowchart Program CGI.....	39
Gambar 3.10 Diagram Aktivitas Program CGI dan Pemetaan HTTP-CoAP .....	40
Gambar 3.11 Diagram Aktivitas <i>Proxy</i> .....	43
Gambar 3.12 Diagram <i>Deployment</i> dari <i>Proxy</i> .....	44
Gambar 3.13 Rancangan Perangkat Keras <i>Proxy</i> .....	45
Gambar 3.14 Rancangan Perangkat Keras CoAP Server.....	46

Gambar 3.15 Skenario Pengujian Sistem.....	47
Gambar 4.1 Pengaturan Jaringan IEEE 802.15.4.....	49
Gambar 4.2 Ping dengan 6LoWPAN.....	50
Gambar 4.3 Skematik dan Desain PCB <i>Proxy</i> .....	50
Gambar 4.4 Hasil Implementasi Perangkat Keras <i>Proxy</i> .....	51
Gambar 4.5 Skematik CoAP Server .....	52
Gambar 4.6 Desain PCB CoAP Server .....	53
Gambar 4.7 Hasil Implementasi Perangkat Keras CoAP Server .....	54
Gambar 4.8 Konfigurasi <i>URL Rewrite</i> .....	55
Gambar 4.9 <i>Pseudocode</i> Program CGI.....	56
Gambar 4.10 <i>Pseudocode</i> Program Pemetaan HTTP-CoAP .....	56
Gambar 4.11 <i>Pseudocode</i> Penanganan Request dalam Satu <i>Thread</i> .....	57
Gambar 4.12 Struktur Data <i>Linked List</i> untuk <i>Cache</i> .....	58
Gambar 4.13 Hasil Pemeriksaan Program <i>Cache</i> dengan Valgrind .....	58
Gambar 4.14 <i>Pseudocode</i> Program CoAP Server.....	59
Gambar 4.15 <i>Pseudocode</i> Program CoAP Server.....	60
Gambar 4.16 Contoh Implementasi URL untuk Meminta Data Sensor .....	61
Gambar 4.17 Contoh Implementasi GET untuk Set GPIO .....	61
Gambar 4.18 Hasil <i>Resource Discovery</i> dalam Bentuk JSON .....	62
Gambar 4.19 Contoh Respon CoAP <i>Multicast</i> .....	62
Gambar 4.20 Contoh Request yang Dilakukan Saat Pengetesan.....	63
Gambar 4.21 Data Respon dalam Apache JMeter .....	64
Gambar 4.22 Jumlah <i>Clients</i> vs <i>Latency</i> .....	64
Gambar 4.23 Jumlah <i>Clients</i> vs <i>Throughput</i> .....	66
Gambar 4.24 Grafik Umur Data <i>Cache</i> vs <i>Latency</i> .....	67
Gambar 4.25 Jumlah <i>Clients</i> vs <i>Error</i> .....	68
Gambar 4.26 Hasil Request yang Bersifat Error Pada Wireshark .....	69

## DAFTAR TABEL

Tabel 2.1 Fitur MRF24J40MA [22].....	26
Tabel 4.1 Pemetaan CoAP <i>Response</i> ke HTTP <i>Response</i> .....	55
Tabel 4.2 Hasil Tes Fungsional.....	62

# BAB 1

## PENDAHULUAN

Bab ini akan memberi informasi mengenai latar belakang, tujuan, batasan masalah, metode penulisan, serta sistematika penulisan dari penelitian ini.

### 1.1 Latar Belakang

*Internet of Things (IoT)* merupakan sebuah topik yang sedang ramai dibicarakan karena kemungkinan dampak yang mungkin muncul akibat dari perkembangan teknologi ini diprediksikan sangat besar. IoT adalah sebuah istilah yang digunakan secara luas untuk suatu kumpulan teknologi, sistem, dan prinsip perancangan yang berhubungan dengan benda-benda terhubung ke Internet yang berdasarkan pada lingkungan fisik [1].

Salah satu visi masa depan dari IoT adalah *Web of Things* [2]. *Web of Things* mengajukan penggunaan standar *web* untuk mengintegrasikan objek-objek (seperti sensor dan aktuator, contohnya) secara penuh ke *World Wide Web* (WWW). Dengan menggunakan teknologi *web*, perkembangan aplikasi dari IoT dapat lebih mudah dan cepat karena mengijinkan *interoperability* dari komunikasi antar alat-alat yang berbeda.

Salah satu contoh bentuk IoT adalah terkoneksiya jaringan sensor nirkabel dengan jaringan Internet dengan standar TCP/IP dan *Web of Things* diraih dengan menerapkan protokol HTTP diatasnya. Namun, jaringan sensor nirkabel tidak bisa begitu saja menerapkan protokol TCP/IP karena jaringan sensor nirkabel berada dalam kategori *constrained network* atau *Low Power and Lossy Networks* (LLN), yaitu jaringan yang memiliki beberapa tantangan seperti lingkungan yang rawan gangguan transmisi, batasan daya yang dimiliki alat karena beroperasi menggunakan baterai, dan juga tingkat keberhasilan pengiriman data yang relatif rendah [3]. Oleh karena itu, dibutuhkanlah standar protokol yang menyerupai standar Internet namun dapat diterapkan pada *constrained networks*. *IPv6 over Low Power Wireless Personal Area Network* (6LoWPAN) dan *Constrained Application Protocol* (CoAP) merupakan contoh protokol pada lapisan jaringan dan lapisan aplikasi yang dibuat untuk *constrained networks*.

Terdapat beberapa keuntungan yang didapat dari menggunakan protokol IP untuk IoT. Keuntungan-keuntungan tersebut adalah piranti dengan basis IP lebih mudah diadaptasikan dengan jaringan internet dengan infrastruktur yang sudah ada, teknologi berbasis IP sudah lama terbukti kinerja dan skalanya, teknologi IP dispesifikasikan dengan cara terbuka dan gratis, dan sudah banyak dikembangkannya alat-alat untuk mengatur dan mendiagnosa jaringan berbasis IP [3].

Untuk dapat terkoneksi dengan dunia internet, jaringan sensor nirkabel membutuhkan *edge router* atau *border router* yang berfungsi sebagai *proxy* untuk komunikasi antara protokol HTTP dan juga CoAP sebagai koordinator jaringan sensor nirkabel, kompresi dan dekompresi *packet header* dari 6LoWPAN, dan mengatur interkoneksi jaringan IPv4/IPv6 [3]–[5].

## 1.2 Tujuan Penelitian

Penelitian skripsi ini memiliki tujuan:

- merancang dan mengimplementasi *border router* untuk jaringan sensor nirkabel,
- merancang dan mengimplementasi perangkat keras dan lunak untuk menerapkan teknologi standar IEEE 802.15.4 pada lapisan fisik dan lapisan MAC, 6LoWPAN pada lapisan jaringan, dan CoAP pada lapisan aplikasi, dan
- merancang dan mengimplementasi sebuah *HTTP-CoAP reverse proxy* yang memiliki *cache* sebagai bagian dari mekanisme *congestion control*.

## 1.3 Batasan Masalah

Penelitian ini bertujuan untuk membuat rancang bangun *border router* sebagai *reverse proxy* untuk jaringan sensor nirkabel dan jaringan Internet. Oleh karena itu, penelitian skripsi ini fokus pada pengembangan lapisan aplikasi dengan bahasan utama:

- translasi dan pemetaan akses dari HTTP ke CoAP dengan metode GET dan
- *caching* pada *cross proxy* sebagai bagian dari *congestion control*.



Selain dengan bahasa utama yang sudah disebutkan sebelumnya, terdapat beberapa batasan dari sistem seperti:

- jaringan bersifat lokal dan
- tidak menggunakan DNS untuk nama proxy dan sensor.

Penelitian skripsi ini tidak mendiskusikan mengenai hal lain seperti konsumsi daya atau keamanan.

#### **1.4 Metode Penelitian**

Metode penelitian mengadopsi *system development life cycle*:

- (1) *requirement analysis*: menentukan spesifikasi sistem baik perangkat keras maupun lunak dengan melakukan studi literatur berdasarkan penelitian serupa,
- (2) *design*: merancang sistem perangkat lunak dan keras yang digunakan,
- (3) *implementation*: memproduksi perangkat keras dan memprogram perangkat lunak yang sudah dirancang,
- (4) *testing*: melakukan pengujian terhadap performa alat dengan *tools* tertentu,
- (5) *evaluation*: analisis hasil pengujian sistem berdasarkan parameter tertentu.

#### **1.5 Sistematika Penulisan**

Dalam penulisannya, buku ini dibagi dalam beberapa bab. Bab 1 menjelaskan mengenai latar belakang, tujuan, batasan masalah, dan metode penelitian. Bab berikutnya, yaitu Bab 2, menjelaskan mengenai dasar teori yang berkaitan dengan penelitian skripsi ini. Pada Bab 3, terdapat informasi mengenai hasil penelitian yang serupa dan pengaruh penelitian tersebut terhadap perancangan serta diagram UML dari perancangan. Bab 4 memberikan hasil dari penelitian, berupa hasil dari tes fungsionalitas dan evaluasi performa. Bab 5 mendiskusikan kesimpulan dari penelitian skripsi ini.

## BAB 2

### TINJAUAN PUSTAKA

Bab ini akan membahas teori dasar yang berhubungan dengan topik penelitian, yaitu mengenai IEEE 802.15.4, 6LoWPAN, CoAP, mekanisme translasi dan *caching* pada *cross-proxy*, *socket programming*, *threading*, dan sekilas mengenai perangkat keras dan lunak yang digunakan.

#### 2.1 Protokol Komunikasi IEEE 802.15.4

Standar IEEE 802.15.4 disahkan pada tahun 2003 dengan tujuan membangun standar yang sederhana dan volume transmisi data yang rendah. Standar ini mendefinisikan *radio frequency* (RF) dan lapisan fisik (PHY) dengan modulasi *Phase-Shift-Key* (PSK) dengan kecepatan 250 kilobit per detik (kbps), beroperasi pada 800, 900, dan 2400 MHz (tergantung pada peraturan lokal suatu negara) dan lapisan *Medium Access Control* (MAC). Masalah transmisi *fading* diatasi dengan *Direct-Sequence Spread Spectrum* (DSSS) pada standar ini. Ada dua metode akses saluran (*channel*) yang digunakan: pertama adalah *Carrier Sense Multiple Access* (CSMA) dengan *Collision Avoidance* (CA), dan yang kedua adalah *Time Domain Multiple Access* (TDMA) menggunakan sinkronisasi *beacon* dan *Guaranteed Time Slots* (GTS) [6].

Jumlah daya output pemancar yang dispesifikasikan sebesar 0,5mW (-3dBm) pada aplikasi jarak pendek (10-50 meter). Sensitivitas daya yang digunakan pada penerima yang dispesifikasikan oleh *Packet Error Rate* (PER). Spesifikasi ini membutuhkan 1% PER pada 85dBm daya saat menerima untuk *frequency band* 2400 MHz.

##### 2.1.1 Pengelompokan Peralatan Dalam Jaringan

Menurut standar IEEE 802.15.4, jaringan *Low-Rate Wireless Private Area Networks* (LR-WPAN) mendukung dua jenis tipe piranti yang berbeda [7, hal. 4]:

- *Full Function Device* (FFD): merupakan piranti yang dapat mendukung tiga mode operasi, dapat melayani sebagai:

- *Personal Area Network (PAN) Coordinator*: merupakan kontroler dari PAN, bertanggung jawab mengenalkan jaringan ke piranti lain yang berhubungan.
- *Coordinator*: menyediakan layanan sinkronisasi melalui transmisi *beacon*. Tidak seperti koordinator PAN, koordinator ini tidak membentuk sebuah jaringan.
- *Simple device*: piranti dalam LR-WPAN yang bukan merupakan dua piranti diatas.
- *Reduced Function Device (RFD)*: merupakan piranti yang beroperasi dengan implementasi minimum dari protocol IEEE 802.15.4. Piranti RFD dimaksudkan untuk aplikasi yang sangat sederhana, seperti *switch* lampu atau sensor infrared yang tidak membutuhkan pengiriman data yang besar.

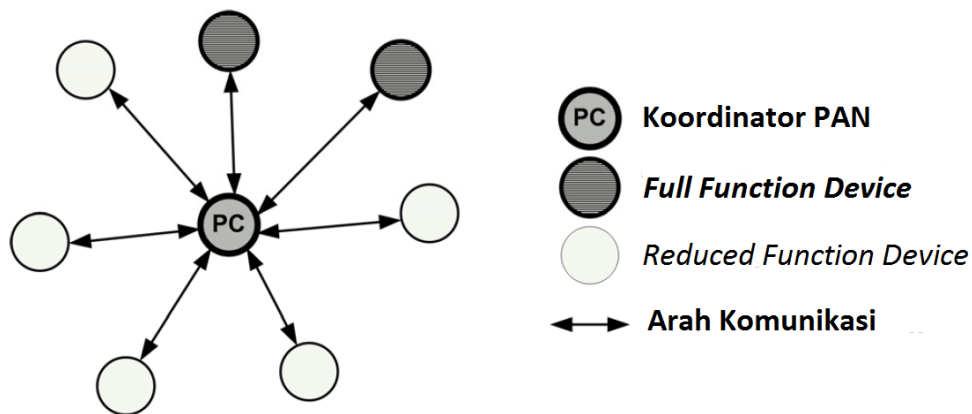
Sebuah LR-WPAN harus memiliki setidaknya satu FFD yang berperan sebagai koordinator PAN yang menyediakan sinkronisasi secara menyeluruh.

#### 2.1.2 Topologi Jaringan IEEE 802.15.4

Ada dua tipe dasar dari topologi jaringan yang didefinisikan oleh IEEE 802.15.4 yang dapat diimplementasikan sesuai dengan kebutuhan aplikasi: topologi bintang (*star*) dan topologi *peer-to-peer* [7].

##### a. Topologi bintang (*star*)

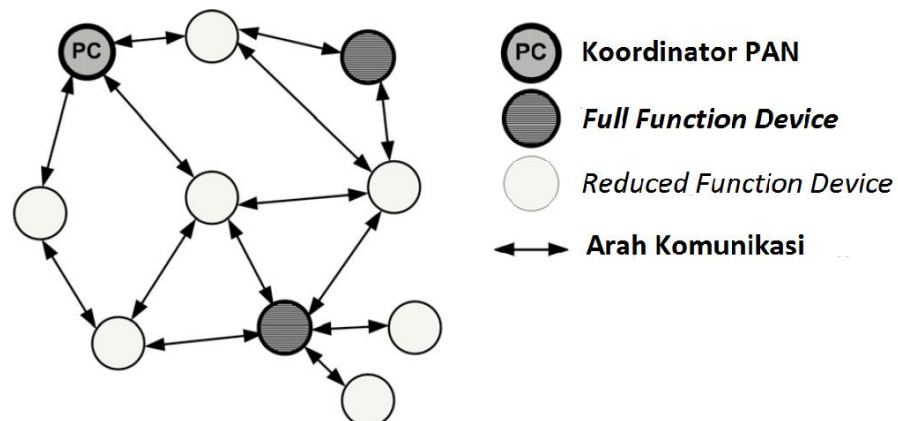
Pada topologi *star*, sebuah node berperan sebagai koordinator PAN dan menjadi pusat dari sebuah jaringan seperti pada Gambar 2.1. Karena kinerja koordinator PAN yang membutuhkan daya yang relatif tinggi, standar IEEE 802.15.4 menyebutkan bahwa koordinator PAN dapat diberi daya utama sedangkan piranti lainnya dapat diberi daya oleh baterai.

Gambar 2.1 Topologi *Star* [7]

Standar IEEE 802.15.4 merekomendasikan topologi ini untuk diaplikasikan pada system otomasi pada rumah, peralatan komputer, dan mainan.

b. Topologi *peer-to-peer*

Topologi *peer-to-peer* juga memiliki koordinator PAN seperti halnya topologi *star*. Namun, paradigmanya berbeda dengan topologi *star*, dimana topologi *peer-to-peer* memiliki paradigma desentralisasi, dimana setiap piranti secara langsung berkomunikasi dengan piranti lain selama masih dalam jangkauan radio. Gambar topologi *peer-to-peer* ditunjukkan pada Gambar 2.2.

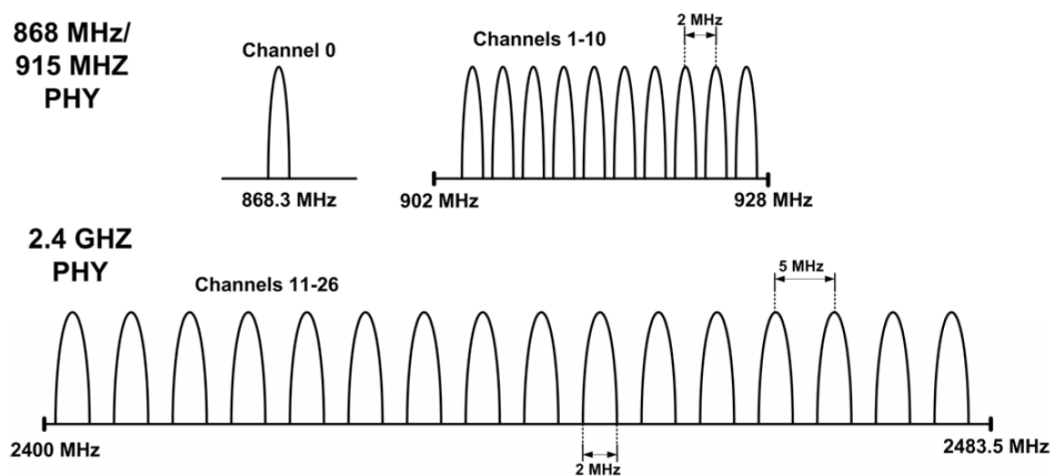
Gambar 2.2 Topologi *Mesh* [7]

Topologi ini memiliki fleksibilitas yang lebih baik dibanding dengan topologi *star*. Pada dasarnya, topologi ini beroperasi secara *ad hoc* dan mengijinkan *routing* data secara *multihop*. Namun kemampuan ini

harus didefinisikan pada lapisan jaringan dan oleh karena itu tidak dianggap bagian dari standar IEEE 802.15.4.

### 2.1.3 Lapisan Fisik IEEE 802.15.4

Lapisan fisik bertanggung jawab untuk transmisi dan penerimaan data menggunakan saluran radio tertentu dan menggunakan teknik modulasi tertentu. IEEE 802.15.4 menawarkan tiga pita frekuensi: 2.4 GHz, 915 Hz, dan 868 MHz. Ada satu saluran diantara 868 dan 868.8 MHz, 10 saluran untuk 902 dan 928 MHz, dan 16 saluran diantara 2.4 dan 2.4835 GHz (Gambar 2.3).



Gambar 2.3 Jumlah *Channel* Untuk Tiap Frekuensi [7]

*Data rate* yang dimiliki adalah 250 kbps pada 2.4 GHz, 40 kbps pada 915 MHz, dan 20 kbps pada 868 MHz [6]. Frekuensi yang lebih rendah lebih cocok untuk transmisi jarak jauh karena tingkat *lost* saat propagasi yang lebih kecil. Transmisi *low rate* menyediakan sensitivitas yang lebih baik dan cakupan area yang lebih besar. *Rate* yang lebih tinggi berarti memiliki *throughput* yang lebih besar, *latency* yang lebih kecil atau *duty cycle* yang lebih kecil.

Lapisan fisik 802.15.4 bertugas mengatur saluran radio dan aliran paket data. *Layer* fisik menggunakan *Carrier Sense Multiple Access* (CSMA) dengan *Collision Avoidance* (CD) untuk mengakses saluran radio. Hal ini berarti bahwa sebuah radio dengan data yang akan ditransmisikan akan “mendengarkan” ke saluran radio terlebih dahulu dan jika saluran tersebut kosong, maka paket akan ditransmisikan. Namun, jika saluran radio tersebut sibuk, entah dikarenakan oleh

stasiun lain sedang mentransmisikan atau karena interferensi, radio akan menahan untuk suatu waktu sebelum mengecek lagi apakah saluran tersebut kosong. IEEE 802.15.4 *MAC layer* mendefinisikan terdapat empat tipe *frame* [3]:

- *Data frame*  
*Frame* yang berisi data sesungguhnya yang ditransmisikan, yang disesuaikan dengan spesifikasi format yang diberikan dari lapisan jaringan.
- *Acknowledgement frame*  
*Frame* yang dimaksudkan untuk dikirim kembali oleh penerima segera setelah penerimaan data frame sebagai penanda bahwa data frame telah diterima.
- *MAC layer command frame*  
*Frame* yang digunakan untuk membolehkan layanan *MAC layer* seperti menghubungkan dan memutuskan hubungan ke koordinator jaringan dan manajemen dari transmisi yang bersifat sinkron.
- *Beacon frame*  
Merupakan *frame* yang digunakan untuk memberikan informasi status jaringan, digunakan oleh koordinator jaringan untuk berkomunikasi dengan *node* yang terhubung.

Semua piranti IEEE 802.15.4 memiliki alamat unik sepanjang 64-bit, yang mirip dengan *MAC address* yang digunakan pada 802.11 *wireless card* atau 802.3 *Ethernet NIC card*. Namun, pada jaringan kompleks yang mentransmisikan data yang kecil, ukuran *header* dikurangi dengan mengijinkan 16-bit alamat lokal.

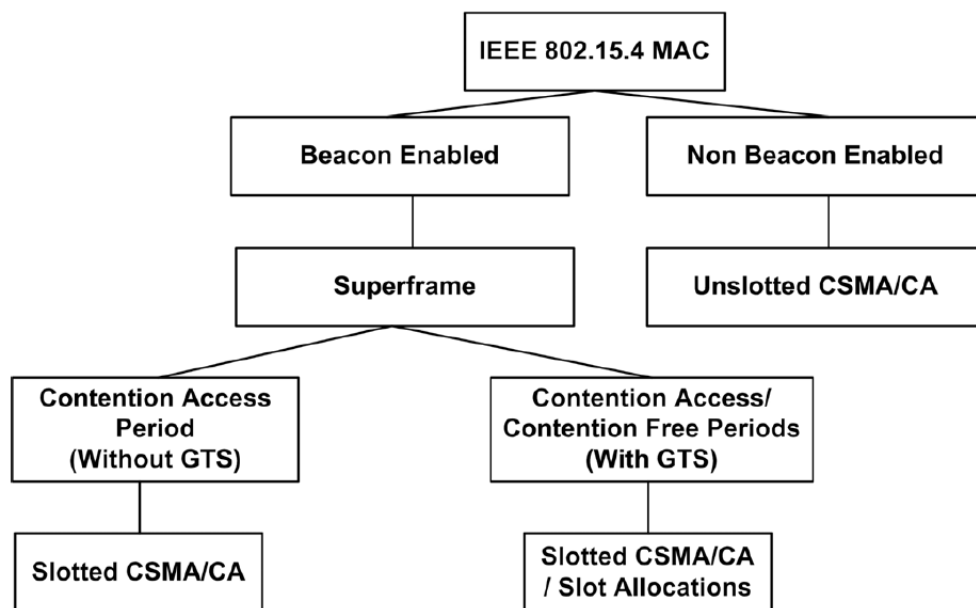
Lapisan fisik mengatur semua symbol dan level bit, juga waktu *switching* pengiriman dan penerimaan paket, dan juga waktu tunda *acknowledgement*. *Layer* fisik dapat dikonfigurasi untuk secara otomatis memberi *acknowledge* untuk setiap paket yang diterima, tergantung pada aplikasi.

#### 2.1.4 Medium Access Control Layer (MAC Layer) IEEE 802.15.4

Sub-lapisan MAC memberikan antarmuka antara lapisan fisik dan lapisan yang lebih atas pada protokol LR-WPAN. Sub-lapisan MAC mendukung dua mode operasional yang bisa didukung oleh coordinator [7]:

- Mode *beacon-enabled*: *beacon* adalah sebuah frame yang bagian dari superframe, yang menanam data frame dari data pertukaran antara *nodes* dan coordinator. Dengan mode ini, *beacon* dihasilkan secara periodik oleh koordinator untuk keperluan sinkronisasi jaringan.
- Mode *non beacon-enabled*: pada mode ini, piranti mengirim data dengan mode CSMA/CA tanpa *superframe*.

Gambar 2.4 menggambarkan struktur dari mode operasi IEEE 802.15.4.



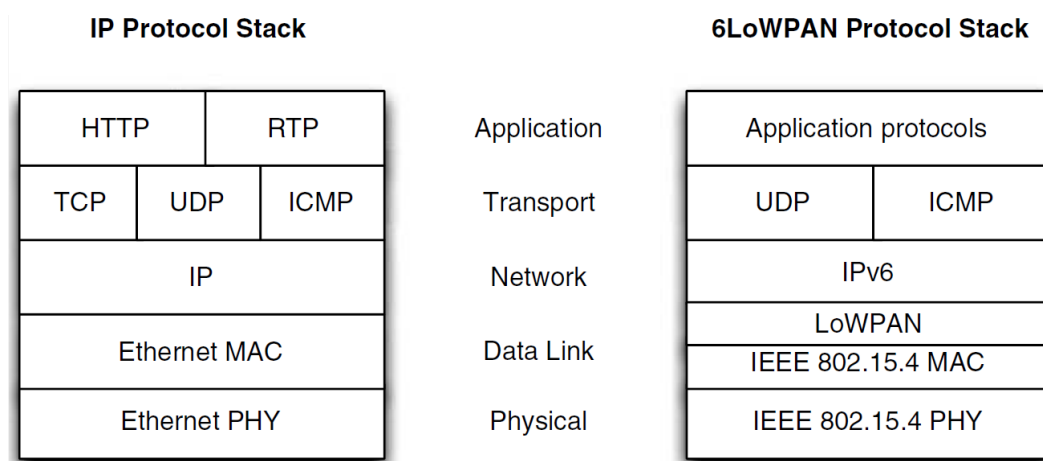
Gambar 2.4 Mode Operasi IEEE 802.15.4 [7]

Lapisan MAC bertanggung jawab untuk menghasilkan *beacon* jaringan yang digunakan untuk mengijinkan piranti menemukan jaringan yang sudah ada, atau untuk kasus jaringan *Time Division Multiple Access* (TDMA), menyediakan indikasi waktu untuk *client* untuk mengakses saluran selama digunakan atau selama saluran tidak digunakan. Tujuan lain dari *beacon* adalah untuk memberi sinyal waktu pada operasi jaringan TDM.

## 2.2 IPv6 over Low Power Wireless Personal Area Networks (6LoWPAN)

*IPv6 Over Low Power Wireless Personal Area Network* (6LoWPAN) adalah sebuah standar yang didefinisikan oleh Internet Engineering Task Force (IETF). Secara teknis, definisi 6LoWPAN adalah sebuah standar yang memungkinkan penggunaan IPv6 secara efisien melalui jaringan nirkabel yang

memiliki keterbatasan seperti rendahnya daya listrik dan tingkat keberhasilan transmisi yang rendah pada sistem tertanam (*embedded systems*) melalui lapisan adaptasi (*adaptation layer*) dan optimisasi protokol-protokol yang terkait [3]. Gambar 2.5 menunjukkan 6LoWPAN *protocol stack* dibandingkan dengan *protocol stack* IP yang umum. Terdapat beberapa perbedaan utama 6LoWPAN dan *protocol stack* IP [3]. Perbedaan pertama adalah 6LoWPAN menggunakan IPv6 dengan sebuah lapisan adaptasi (disebut juga *LoWPAN adaptation layer*) agar dapat ditransmisikan diatas IEEE 802.15.4. Perbedaan kedua adalah 6LoWPAN menggunakan *User Datagram Protocol* (UDP), sedangkan IPv4/IPv6 menggunakan *Transmission Control Protocol* (TCP).



Gambar 2.5 IP dan 6LoWPAN Protocol Stacks [3]

Terdapat beberapa keuntungan dari penggunaan Internet Protocol (IP) pada *Internet of Things*, yaitu [3]:

- peralatan (piranti) berbasis IP dapat terhubung dengan relatif mudah dengan jaringan IP lain,
- jaringan IP memungkinkan penggunaan infrastruktur jaringan yang sudah ada,
- teknologi berbasis IP sudah terbukti bekerja dengan baik,
- teknologi IP dispesifikasikan dengan terbuka, dengan proses pembentukan standard an dokumen yang dapat diakses oleh semua orang,



- alat-alat untuk mengatur, menugaskan, dan mendiagnosa jaringan berbasis IP sudah tersedia.

Dikarenakan jumlah alamat IP dari IPv4 diprediksi akan tidak mencukupi untuk memenuhi kebutuhan masa depan (sebanyak  $2^{32}$ , atau 4.294.967.296), IETF pada tahun 1998 mengeluarkan versi IP yang lebih baru, yaitu IPv6 yang dapat memiliki jumlah IP lebih banyak, yaitu  $2^{128}$  atau sebanyak  $3.4 \times 10^{38}$ . Namun, IPv6 tidak cocok untuk aplikasi jaringan sensor nirkabel yang berdaya rendah yang berdiri diatas IEEE 802.15.4 karena beberapa alasan [8]:

- IPv6 memiliki *Maximum Transmission Unit* (MTU) sebesar 1280 byte, sedangkan MTU untuk IEEE 802.15.4 adalah 127 byte,
- kelebihan data pada IPv6 *header* dianggap merupakan suatu pengeluaran tambahan yang signifikan pada jaringan sensor nirkabel,
- mekanisme dari IPv6 membutuhkan banyak proses, yang tidak cocok untuk *constrained devices* pada jaringan sensor nirkabel, dan
- pendekatan *routing* untuk topologi *mesh* tidak didefinisikan pada protokol IPv6

6LoWPAN bukanlah satu-satunya protokol pada lapisan jaringan yang dapat diterapkan untuk jaringan sensor nirkabel. Selain 6LoWPAN, contoh protokol lainnya yang populer adalah ZigBee dan WirelessHART. Penggunaan 6LoWPAN secara ideal adalah saat [3]:

- piranti sistem tertanam (*embedded devices*) butuh berkomunikasi dengan layanan berbasis internet,
- jaringan yang bersifat heterogen membutuhkan ikatan agar dapat saling berkomunikasi,
- jaringan perlu terbuka, terpakai, dan tersedia untuk pengguna baru,
- skalabilitas dibutuhkan untuk melintasi infrastruktur jaringan.

Format 6LoWPAN mendefinisikan cara komunikasi IPv6 melalui IEEE 802.15.4 frame dan menspesifikasikan elemen kunci dari *adaptation layer*. 6LoWPAN memiliki tiga elemen utama [9]:

- *Header compression*.

IPv6 header dikompres dengan membuang yang dianggap tidak perlu.

- *Fragmentation*

Paket IPv6 difragmentasi menjadi beberapa frame agar dapat ditransmisi melalui IEEE 802.15.4 mengingat maksimum unit transmisi yang diijinkan.

- *Layer-2 forwarding*

Untuk mendukung meneruskan datagram IPv6.

### 2.2.1 Arsitektur 6LoWPAN

Jaringan 6LoWPAN pada proyek skripsi ini dibuat dengan menghubungkan beberapa piranti *embedded*, dimana jaringan ini merupakan *stub network* pada internet, yaitu jaringan dimana paket IP dikirimkan dan diterima tanpa meneruskan ke jaringan lain. Jaringan ini disebut dengan *low-power wireless area networks* (LoWPAN), dan terhubung ke jaringan IP lain dengan menggunakan *border router* yang berperan sebagai *gateway*. Sebuah LoWPAN terdiri dari *nodes* (sistem mikrokontroler + sensor) yang berfungsi sebagai server *border router*. *Border router* memiliki peran penting dalam LoWPAN karena bertanggung jawab dalam menyalurkan paket data ke dalam dan keluar LoWPAN, melakukan kompresi 6LoWPAN, dan melakukan *neighbor discovery* dalam LoWPAN. Ada tiga jenis arsitektur LoWPAN seperti yang digambarkan pada Gambar 2.6:

- *Simple LoWPAN*

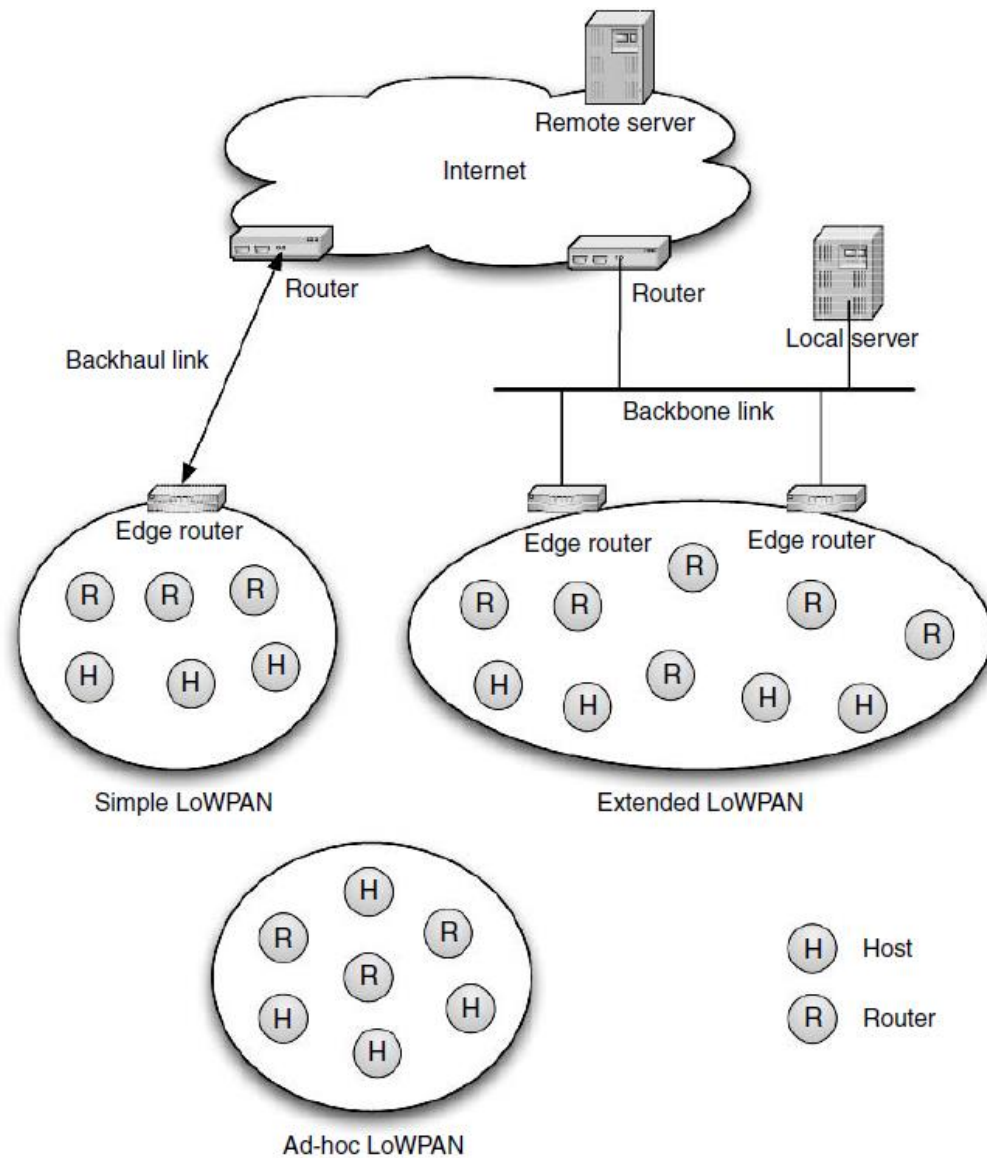
*Simple LoWPAN* adalah jenis arsitektur LoWPAN yang terhubung ke jaringan lain melalui satu *border router*

- *Extended LoWPAN*

*Extended LoWPAN* adalah LoWPAN yang terdiri dari beberapa LoWPAN yang terhubung dengan beberapa *border router* sepanjang *backbone link* (misalnya Ethernet)

- *Ad hoc LoWPAN*

*Ad hoc LoWPAN* adalah LoWPAN yang tidak terhubung ke Internet.



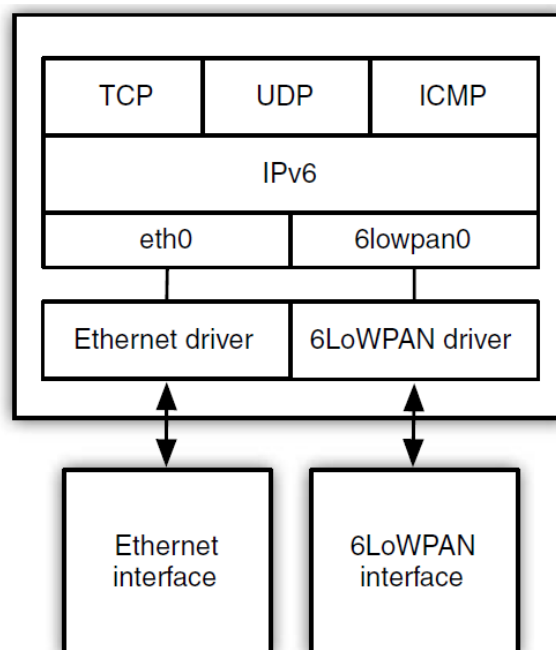
Gambar 2.6 Arsitektur 6LoWPAN [3]

### 2.2.2 Integrasi Jaringan IP dengan 6LoWPAN dengan *Border Router*

Agar jaringan 6LoWPAN dapat terhubung dengan jaringan IP lainnya, digunakanlah *border router* sebagai penengah. Pada umumnya, *border router* membutuhkan [3]:

- antarmuka nirkabel 6LoWPAN,
- lapisan adaptasi 6LoWPAN,
- 6LoWPAN *Neighbor Discovery*, dan
- IPv6 atau IPv4 *protocol stack*.

Mengintegrasikan 6LoWPAN dalam *border router* berbeda arsitekturnya dengan mengintegrasikan 6LoWPAN dalam *node*. Ada beberapa alasan mengapa *border router* pengimplementasiannya berbeda dengan *node*. Pertama, karena *border router* terhubung dengan jaringan berbasis IP, *border router* sudah memiliki standar *protocol stack* untuk jaringan IP. Kedua, *border router* memiliki beberapa fungsi tambahan seperti adanya adaptasi antara IPv6 dan 6LoWPAN. Terakhir, *border router* juga bertugas untuk *routing* data 6LoWPAN dan IP dengan mekanisme *proxy*. *Border router* biasa diimplementasikan menggunakan Linux sebagai system operasinya. Sebuah arsitektur *border router* berbasis UNIX ditunjukkan pada Gambar 2.7.



Gambar 2.7 *Border Router* dengan Antarmuka 6LoWPAN [3]

Salah satu cara untuk mengintegrasikan 6LoWPAN ke *border router* adalah dengan menyediakan fungsi lapisan 1-3 (fisik, MAC, dan jaringan) dalam prosesor. Antarmuka 6LoWPAN dapat direalisasikan dengan menggunakan modul radio dengan UART, SPI, atau USB.

Untuk mendukung antarmuka nirkabel, dibutuhkan sebuah *driver* pada system operasi yang ditanamkan ke *border router*. *Driver* ini mengimplementasikan antarmuka ke 6LoWPAN *stack* pada antarmuka nirkabel. Agar dapat meneruskan

data dari jaringan 6LoWPAN ke IP biasanya *driver* mengemulasikan antarmuka jaringan dalam system operasi seperti contohnya 6LoWPAN pada *protocol stack* yang berbasis Unix. Metode antarmuka jaringan seperti ini didukung oleh Contiki uIP, TinyOS BLIP dan NanoStack Linux support.

Untuk menggunakan antarmuka nirkabel 6LoWPAN dengan standar IPv6 *protocol stack*, fungsi-fungsi berikut perlu diimplementasikan.

- Lapisan adaptasi LoWPAN: 6LoWPAN *frame* yang diterima dari *link-layer* perlu disatukan (*decompressed*) seperti yang dispesifikasikan pada standar RFC 4944. Pada arah sebaliknya, *frame* IPv6 perlu untuk dikompres. Langkah ini dapat dilakukan dalam antarmuka nirkabel atau *driver* pada *border router*.
- 6LoWPAN *Neighbor Discovery* (6LoWPAN-ND): 6LoWPAN-ND melakukan *Router Advertisement* (RA) untuk memberitahukan *node* baru untuk bergabung ke jaringan yang ada seperti yang dispesifikasikan pada RFC 4861.

Secara ideal, *border router* memiliki fitur-fitur sebagai berikut.

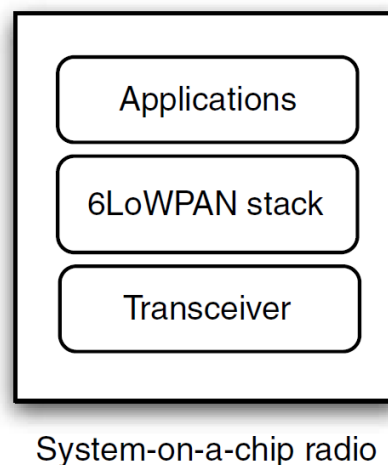
- IPv6 *routing*: Mengurus masalah *routing* paket dalam dan keluar dari LoWPAN.
- Interkoneksi IPv6/IPv4: Walaupun IPv6 mulai populer, IPv4 masih digunakan secara luas. Oleh karena itu interkoneksi IPv6/IPv4 diperlukan. Metode yang umum digunakan adalah *tunneling* IPv6-in-IPv4 (dikenal juga dengan 6in4) dan *automatic tunneling* (dikenal juga dengan 6to4).
- Firewall dan *access control*: Firewall bertugas untuk melindungi jaringan dari *malicious traffic*, sedangkan *access control* memberikan layanan jaringan hanya pada *node* yang berhak.
- Manajemen jaringan: *Border router* merupakan tempat yang ideal untuk memberikan manajemen LoWPAN dengan protokol standar seperti SNMP.
- *Proxy*: *Application protocol proxy* seperti *proxy* untuk *cross-protocol* dibutuhkan agar klien dapat menghubungi dan mengambil *resource* dari server walaupun berbeda basis protokol.

### 2.2.3 Mengimplementasikan 6LoWPAN

Untuk mengimplementasikan 6LoWPAN, dibutuhkan piranti yang sudah diintegrasikan dengan modul 6LoWPAN baik dalam bentuk perangkat keras dalam *chip* maupun perangkat lunak untuk *protocol stack*.

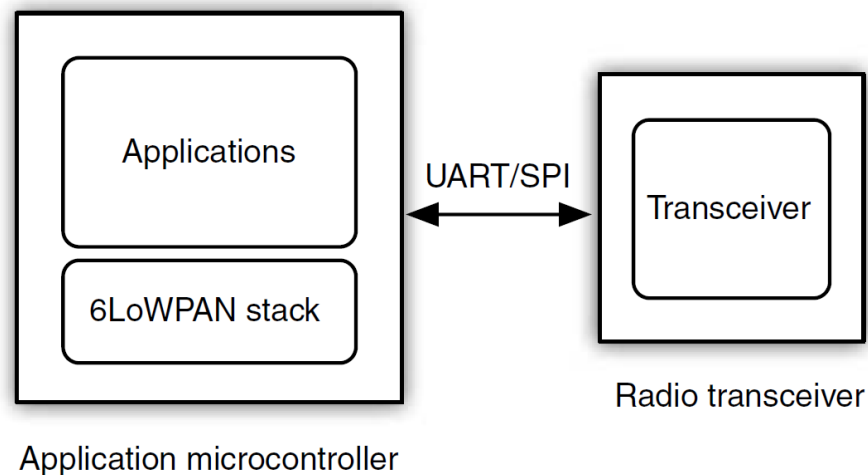
#### a. 6LoWPAN dalam Chip

Karena 6LoWPAN adalah teknologi jaringan yang digunakan pada alat *embedded*, 6LoWPAN *protocol stack* ditanamkan ke alat *embedded* seperti mikrokontroler. Ada tiga model berbeda dari *chip* yang digunakan: *chip* tunggal, *chip* ganda, dan *network processor*. *Chip* tunggal menggunakan radio *System-on-a-Chip* (SoC) dengan mikrokontroler yang sudah terintegrasi, sehingga *protocol stack*nya sudah tertanam dalam satu modul, seperti yang digambarkan pada Gambar 2.8. Contoh dari jenis SoC adalah TI CC2530, TI CC1110, dan Jennic JN5139.

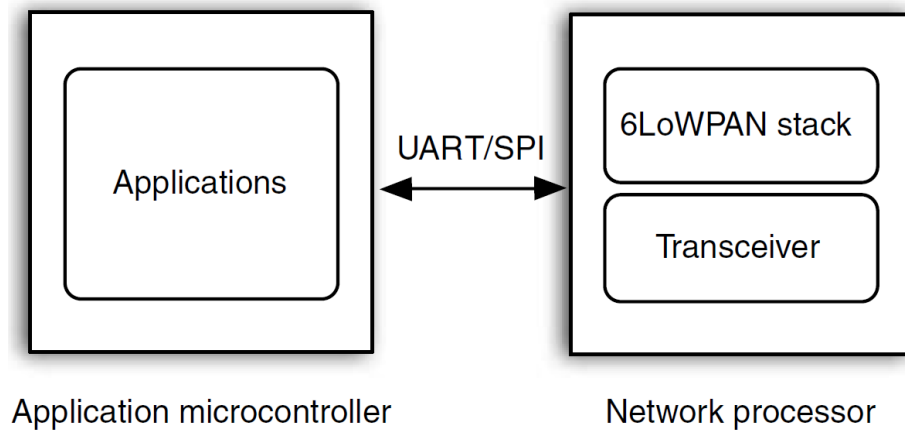


Gambar 2.8 Model *System-on-a-chip* [3]

Bentuk teknologi *chip* ganda menggunakan mikrokontroler yang terpisah dengan *transceiver*, seperti pada Gambar 2.9. Biasanya mikrokontroler atau prosesor mengakses *transceiver* dengan *universal asynchronous receiver/transmitter* (UART) atau dengan *serial programming interface* (SPI). Contoh dari *transceiver* jenis ini adalah TI CC2520, MRF24J40MA, dan Atmel AT86RF231.

Gambar 2.9 Model *dual-chip* [3]

Solusi 6LoWPAN dengan *network processor* adalah dengan menggunakan radio *transceiver* yang memiliki *protocol stack* namun menggunakan prosesor yang terpisah seperti yang ditunjukkan pada Gambar 2.10. Komunikasi ke *network processor* dapat menggunakan UART atau SPI. Model ini sering digunakan untuk *border router* karena mudah diintegrasikan dengan Linux.

Gambar 2.10 Model *Network Processor* [3]

#### b. Protocol Stack

Salah satu cara untuk menggunakan 6LoWPAN adalah dengan mengintegrasikan *protocol stack* yang sudah ada. *Protocol stack* untuk 6LoWPAN memiliki komponen:

- radio driver,
- medium access control (MAC),
- IPv6 dengan 6LoWPAN,
- UDP,
- ICMPv6,
- Neighbor Discovery,
- API ke *protocol stack*.

Beberapa cara dapat digunakan untuk memperoleh *protocol stack* 6LoWPAN pada sistem operasi, misalnya dengan uIPv6 untuk Contiki dan BLIP untuk TinyOS.

### 2.3 *Constrained Application Protocol (CoAP)*

CoAP merupakan sebuah protokol pada lapisan aplikasi yang fokus pada komunikasi *machine-to-machine* (M2M) pada sebuah jaringan yang memiliki keterbatasan dalam hal ukuran paket yang dapat dikirim, *latency* yang tinggi, *loss rate* pengiriman paket yang relatif signifikan, dan kemungkinan memiliki *bandwidth* yang sempit [10]. CoAP berdiri diatas arsitektur *Representational State Transfer* (REST) seperti halnya HTTP, dengan tujuan sebagai protokol komunikasi pada lapisan aplikasi untuk menggantikan HTTP pada jaringan sensor nirkabel. Terdapat beberapa alasan mengapa protokol HTTP tidak cocok digunakan pada *constrained networks* [11]:

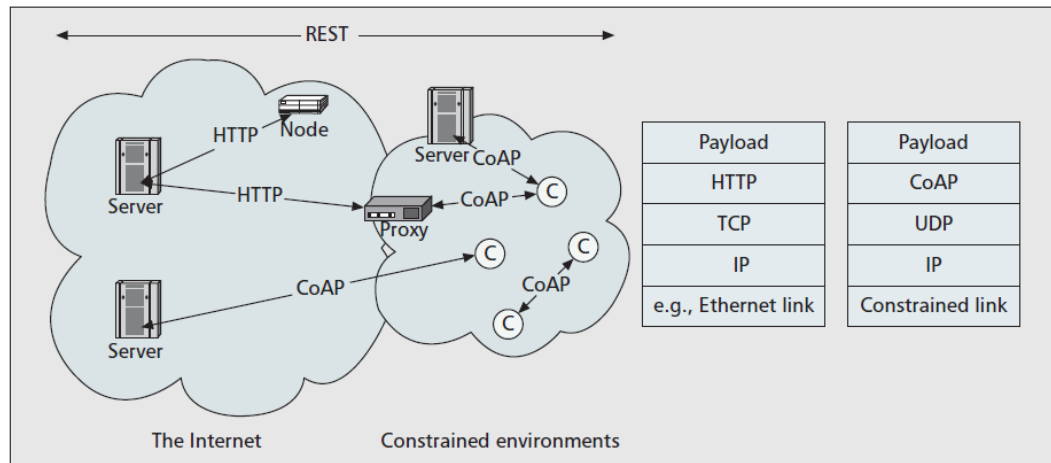
- HTTP merupakan protokol yang sifatnya *human-readable*, artinya HTTP merepresentasikan data berupa teks ASCII yang dapat dibaca manusia yang artinya tidak efisien, tidak seperti CoAP yang bersifat *binary protocol*,
- HTTP menggunakan *Transmission Control Protocol* (TCP) yang bersifat *connection-oriented*, sedangkan CoAP menggunakan *User Datagram Protocol* (UDP) yang bersifat *connectionless*.

CoAP dimaksudkan untuk menjadi semacam alternatif untuk HTTP pada *constrained networks*. Perbandingan *protocol stack* HTTP dan CoAP diberikan pada Gambar 2.11. Untuk merealisasikan arsitektur *web* pada jaringan



*constrained networks* dengan komunikasi M2M, CoAP memiliki beberapa fitur [12]:

- *Header yang compact*  
CoAP memiliki ukuran *base header* sebesar 4 byte, dan total *header* hanya sebesar 10-20 byte
- Metode dan *Uniform Resource Identifier* (URI)  
Agar *client* dapat mengakses *resource* di server, CoAP mendukung metode *request* seperti GET, PUT, POST, dan DELETE. CoAP mendukung fitur URI, yang mana merupakan kunci dari arsitektur *web*
- Tipe konten  
CoAP memungkinkan secara eksplisit mengindikasikan tipe konten di *header* dari *payload* yang ditransmisikan
- *Caching* yang sederhana  
*Caching* diperlukan untuk mengoptimasikan performa pada *constrained networks*
- Pengikatan paket pada lapisan *transport*  
CoAP menggunakan UDP yang lebih hemat *resource* dibandingkan dengan TCP
- *Resource discovery*  
Digunakan untuk menemukan daftar *resource* yang ditawarkan pada piranti di jaringan
- *Subscription*  
Salah satu kelemahan dari HTTP adalah kebergantungannya pada model *pull* untuk mendapatkan data. CoAP menggunakan pendekatan secara asinkron untuk mendukung *push* informasi dari server ke *client*.



Gambar 2.11 *Constrained RESTful Environment architecture* [12]

Untuk dapat menjelajah jaringan sensor nirkabel dengan protokol CoAP, *translating proxy* dibutuhkan untuk membangun koneksi dengan dunia *web* dengan cara mengizinkan *HTTP client* meminta *resource* dari server. Subbab berikutnya akan membahas mengenai *proxy* pada *constrained networks* yang ingin diteliti pada skripsi ini.

### 2.3.1 Cross-Protocol Proxy

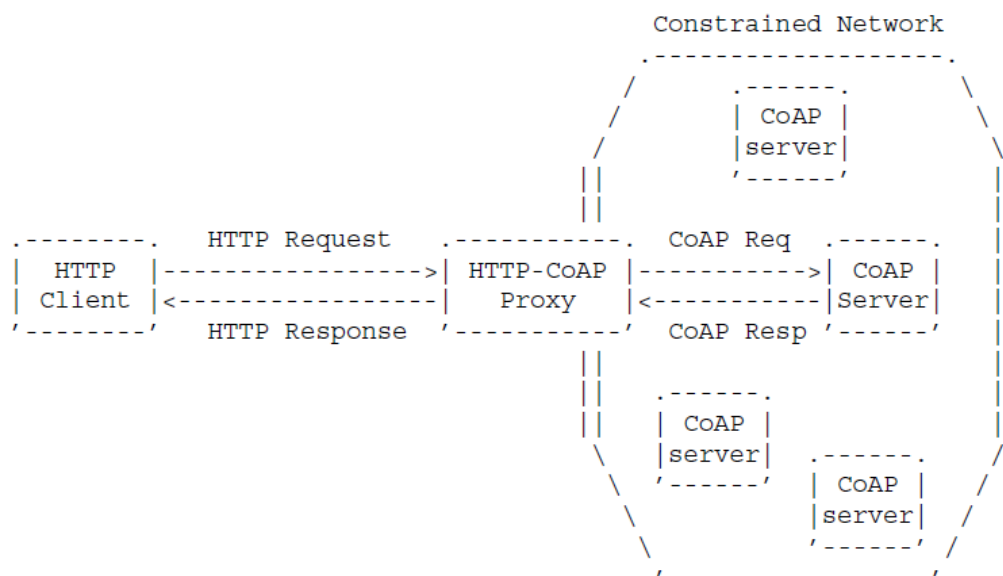
*Proxy* adalah sebuah penengah yang tugas utamanya adalah meneruskan permintaan (*forwarding request*) dan menyampaikan kembali respon dari server, memungkinkan adanya *caching*, atau penerjemahan/translasi protokol pada prosesnya [13]. Pada kasus ini, *proxy* yang dimaksud dengan *proxy* adalah *cross-protocol proxy*, yaitu sebuah *proxy* yang menerjemahkan protokol yang berbeda, seperti CoAP ke HTTP dan HTTP ke CoAP. Ada dua jenis bentuk *proxy* yang umum [13] yang dipaparkan sebagai berikut.

- *Forward-proxy*: sebuah *endpoint* dari jaringan CoAP yang dipilih oleh *client*, biasanya dengan cara konfigurasi secara local pada *client*, untuk melakukan permintaan (*request*) sesuai dengan kepentingan *client*, dan menerjemahkan protokol jika hanya dibutuhkan.
- *Reverse-proxy*: sebuah *endpoint* dari jaringan CoAP yang berdiri untuk merepresentasikan satu atau lebih server. Sebuah *reverse-proxy* menerima

permintaan (*request*) seolah *proxy* ini adalah target server yang ingin diperoleh datanya.

*Proxy* berguna pada *constrained networks* untuk membatasi *network traffic*, meningkatkan performa jaringan, memungkinkan teknik meminta *resource* dari node yang dalam kondisi *sleep*, dan menangani masalah keamanan. Pada penelitian skripsi ini, penulis akan merancang sebuah *reverse-proxy* dengan standar yang sudah ditetapkan oleh IETF.

Sebuah *reverse HTTP-CoAP (HC) proxy* adalah *proxy* yang dapat diakses oleh *client* yang hanya memiliki koneksi dengan protokol HTTP, memetakan HTTP *request* menjadi CoAP *request*, lalu meneruskan ke server berbasis CoAP, yaitu *sensor node*. Gambar 2.12 menggambarkan skenario *deployment* dari *reverse proxy* ini.



Gambar 2.12 Skenario HTTP-CoAP *reverse proxy* [14]

### 2.3.2 URI Mapping

Hingga penelitian skripsi ini ditulis, status petunjuk pemetaan HTTP-CoAP masih berstatus *draft* yang akan kadaluarsa pada 14 November 2016 [14].

*URI Mapping* (pemetaan URI) adalah sebuah proses dimana URI dari CoAP *server* ditransformasi menjadi HTTP URI sehingga:

- *client* yang hanya mendukung HTTP dapat melakukan permintaan,
- *proxy* yang menerima permintaan dapat mengekstrak CoAP URI yang dimaksud tanpa ambigu.

*Default mapping* dilakukan dengan cara menempelkan CoAP URI dalam bentuk HTTP URI. Contohnya, jika URI dari HC *proxy* adalah `http://p.example.com/hc` dan target CoAP URI adalah `coap://s.example.com/light`, maka hasil dari HTTP URI yang digunakan menjadi

`http://p.example.com/hc/coap://s.example.com/light`.

Dengan *default mapping*, komponen *query* pada URI dimasukkan ke dalam HTTP URI seperti contohnya jika CoAP URI dengan *query* berbentuk `coap://s.example.com/light?dim=5` menjadi `http://p.example.com/hc/coap://s.example.com/light?dim=5`.

### 2.3.3 Caching

*Caching* adalah sebuah mekanisme yang digunakan sebagai teknik kontrol kemacetan jaringan. Dengan *caching*, server tidak harus mengirim ulang *request* yang sejenis dalam rentang waktu tertentu sehingga *network traffic* yang kurang perlu dapat dihindari. Dalam beberapa kasus, respon dari *request* sebelumnya dapat digunakan kembali untuk *request* mendatang, sehingga dapat menurunkan *lantency* (interval antara waktu *request* dengan respon); mekanisme “*freshness*” digunakan untuk tujuan ini. Bahkan saat *request* baru diutuhkan, penggunaan kembali *payload* dari respon sebelumnya untuk menjawab *request* dimungkinkan, sehingga mengurangi penggunaan *bandwidth*; mekanisme “*validasi*” digunakan untuk tujuan ini. Dua model mekanisme *caching* diatas adalah *Freshness Model* dan *Validation Model* [13].

#### a. *Freshness Model*

Saat *endpoint* memiliki satu atau lebih respon yang disimpan untuk permintaan GET, tetapi tidak bisa menggunakan satupun dari respon yang tersimpan (misalnya karena respon-respon tersebut dianggap sudah tidak “*fresh*”), *endpoint* dapat menggunakan *ETag Option* dalam *CoAP Option* di dalam

permintaan GET untuk memberikan server kesempatan untuk memilih respon yang tersimpan yang akan digunakan dan memperbarui “kesegaran” dengan memperbarui data. Proses ini disebut dengan validasi atau revalidasi respon yang disimpan di *cache*.

CoAP *endpoint* diizinkan untuk menggunakan respon yang tersimpan dalam *cache* saat:

- metode *request* yang digunakan untuk mengambil *resource* sesuai dengan respon yang disimpan,
- semua pilihan (CoAP Option) yang ada di *request* sesuai kecuali ada pilihan “NoCacheKey”, dan
- respon yang disimpan “fresh” atau “valid”.

#### *b. Validation Model*

Ada beberapa alasan dari pembuatan *proxy* antara CoAP dan HTTP, contohnya saat mendesain sebuah halaman *web* untuk digunakan di salah satu protokol tersebut. Ada dua arah yang mungkin untuk mengakses *resource* melalui *forward-proxy*:

- CoAP-HTTP *Proxying*: memungkinkan klien CoAP mengakses *resource* pada HTTP server melalui sebuah proses/piranti penengah. Hal ini diawali dengan memasukkan *Proxy-Uri* atau pilihan skema *proxy* dengan “http” atau “https” dalam *request* CoAP ke CoAP-HTTP *proxy*.
- HTTP-CoAP *proxying*: memungkinkan klien HTTP mengakses *resource* pada CoAP server melalui sebuah proses/piranti penengah. Hal ini diawali dengan memasukkan “coap” atau “coaps” di *request* HTTP ke HTTP-CoAP *proxy*.

## **2.4 Socket Programming**

*Socket* adalah sebuah file (*file descriptor*) yang digunakan sebagai sebuah titik untuk berkomunikasi dengan proses lain melalui jaringan (atau dalam suatu sistem operasi yang sama, namun berbeda proses) [15]. Pada penelitian skripsi ini digunakan dua tipe *socket*, yaitu *Internet socket* dan *UNIX domain socket*.

Ada dua jenis dari *Internet socket*, yang pertama adalah *Stream Socket* dan yang kedua adalah *Datagram Socket*. *Stream Socket* adalah jenis *socket* yang digunakan untuk komunikasi yang bersifat *reliable*, seperti yang ditemukan pada komunikasi berbasis TCP/IP. *Datagram socket* digunakan untuk komunikasi yang bersifat *connectionless*, seperti yang ditemukan pada komunikasi berbasis UDP [16].

*UNIX domain socket* adalah suatu bentuk file untuk komunikasi inter proses (*Inter Process Communication* atau disingkat IPC), yang memungkinkan komunikasi antar proses yang berjalan pada mesin yang sama [17].

Pada saat memprogram dengan *socket*, umumnya dibentuk model *client-server*, dimana *client* bertindak sebagai yang meminta data ataupun memberi data dan bersifat aktif, sedangkan *server* menunggu koneksi dari *client* secara pasif. Pada penelitian ini, karena sistem operasi yang digunakan berbasis Linux (keluarga Debian), maka yang digunakan adalah *POSIX socket*, bukan *WinSock* yang umumnya digunakan dalam sistem operasi Windows.

## 2.5 Threading

*Thread* adalah sebuah istilah untuk suatu bagian dari perangkat lunak yang dapat mengerjakan sesuatu dalam komputer. Dengan *thread*, *programmer* dapat membuat aplikasi yang menggunakan *resource* sistem dengan lebih efisien dan berjalan dengan cepat pada komputer berbasis multiprosesor [18]. Model *thread* yang digunakan adalah “*Pthread*”, atau “*POSIX thread*” yang dapat ditemukan di sistem operasi yang *POSIX-compliant*. Aplikasi dapat dikatakan bersifat *multithread* jika dalam proses berjalannya menjalankan beberapa (lebih dari 1) *thread*. Hal ini sangat berguna untuk beberapa jenis aplikasi tertentu, misalnya dalam aplikasi server yang menerapkan *socket programming* agar dapat menangani beberapa *client* secara *concurrent*.

Permasalahan dalam menerapkan aplikasi *multithreading* akan muncul jika tiap *thread* diijinkan melakukan aksi berupa pembacaan, penulisan, dan penghapusan pada suatu data yang bersifat global tanpa suatu kebijakan yang jelas. Hal ini dapat memicu hasil data yang tidak konsisten untuk tiap *thread* yang

berusaha mengakses data tersebut, menjadikan program yang dibuat bersifat *non-deterministic*. Terdapat beberapa cara untuk mengatasi masalah ini, namun pada penelitian ini yang digunakan adalah *mutual exclusion* atau juga biasa disebut *mutex*. *Mutex* berusaha mengunci *resource* atau data yang dapat diakses oleh *thread* lain dan melepaskan kunci tersebut ke *thread* lain yang memintanya jika sudah selesai dilakukan operasi pada data global atau *resource* tersebut. Proses dimana terjadi pengaksesan data global tersebut dinamakan *critical section*.

Sebagai gambaran bagaimana *mutex* bekerja, jika ada *thread* “a” berusaha mengunci saat *thread* “b” sedangkan mengunci dengan kunci yang sama, maka *thread* “a” akan “tidur” hingga *thread* “b” melepaskan kunci tersebut. Begitu seterusnya jika terdapat lebih dari dua *thread* [19].

## 2.6 Perangkat Keras dan Lunak yang Digunakan

Dalam skripsi ini, digunakan tiga perangkat keras utama, yaitu komputer papan tunggal (*Single Board Computer*) Raspberry Pi Model B rev 2, Wi-Fi Dongle USB adapter TP-LINK TL-WN725N, dan modul radio MRF24J40MA.

### 2.6.1 Raspberry Pi Model B rev 2

Seri Raspberry Pi adalah *Personal Computer* (PC) yang berukuran kecil namun memiliki kelebihan dibanding komputer biasa yaitu dapat berinteraksi dengan dunia luar dengan *port* yang dapat disambungkan dengan alat-alat tambahan seperti sensor, actuator dan sebagainya. Detail teknis dari Raspberry Pi Model B rev 2 [20]:

- 700 MHz ARM11 ARM1176JZF-S core
- RAM 512MB
- 2 USB *ports*
- 26 GPIO *pins*
- Full HDMI *port*
- 3.5mm audio *jack*
- Konsumsi daya 3.0W
- *Display interface* (DSI)
- Micro SD card *slot*

SBC yang menggunakan sistem operasi Linux memiliki keuntungan sebagai berikut [21]:

- *System driver* yang dapat digunakan kembali untuk jaringan, penyimpanan, dan *peripheral* lain
- Perangkat lunak tambahan yang jumlahnya banyak dan secara aktif dikembangkan karena bersifat *open source*
- Peralatan *gateway* berbasis Linux dapat dimonitor dengan infrastruktur yang sudah umum atau lazim
- Linux merupakan sistem operasi yang relatif populer sehingga administrator relatif lebih mudah menguasai
- Fleksibel karena perangkat keras dapat ditukar dengan yang lebih baru (*upgrade*)

#### 2.6.2 Wi-Fi Dongle USB Adapter

Jenis Wifi Dongle yang digunakan adalah TP-LINK TL-WN725N yang dapat digunakan untuk menghubungkan komputer ke jaringan nirkabel Wi-Fi dengan standar IEEE 802.11 dengan kecepatan maksimum 150Mbps.

#### 2.6.3 Modul Radio MRF24J40MA

MRF24J40MA adalah modul radio IEEE 802.15.4 yang beroperasi pada frekuensi 2.4GHz. MRF24J40MA memiliki antenna PCB yang sudah terintegrasi. Bentuk fisik dari modul ini dapat dilihat pada Gambar 2.15 dan fitur dari *transceiver* ini terlihat pada Tabel 2.1.

Tabel 2.1 Fitur MRF24J40MA [22]

Nama Parameter	Nilai
Tipe	Modul IEEE 802.15.4
Keluarga	Nirkabel
<i>Clock</i>	20 MHz
Antarmuka	4-wire SPI
Jumlah Pin	12
RF Module	Ya
RF Transceiver	Ya

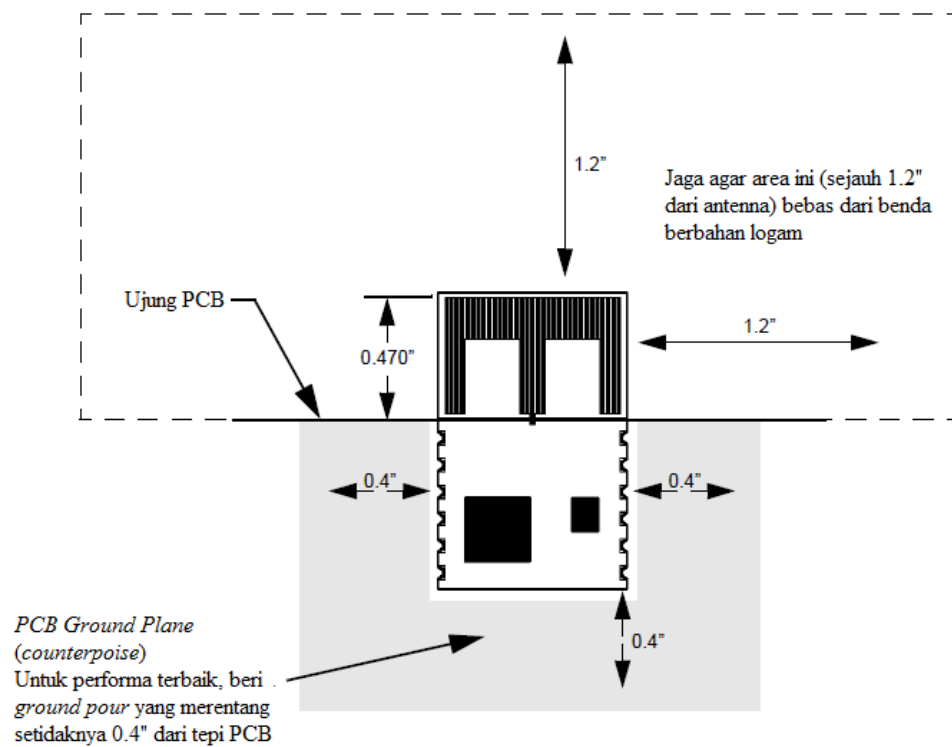


Suhu Operasi	-40° C hingga +85° C
Tegangan Operasi	2.4 - 3.6 V
Konsumsi arus Tx	23 mA
Konsumsi arus Rx	19 mA
<i>Sleep</i>	2 $\mu$ A
Enkripsi	AES128



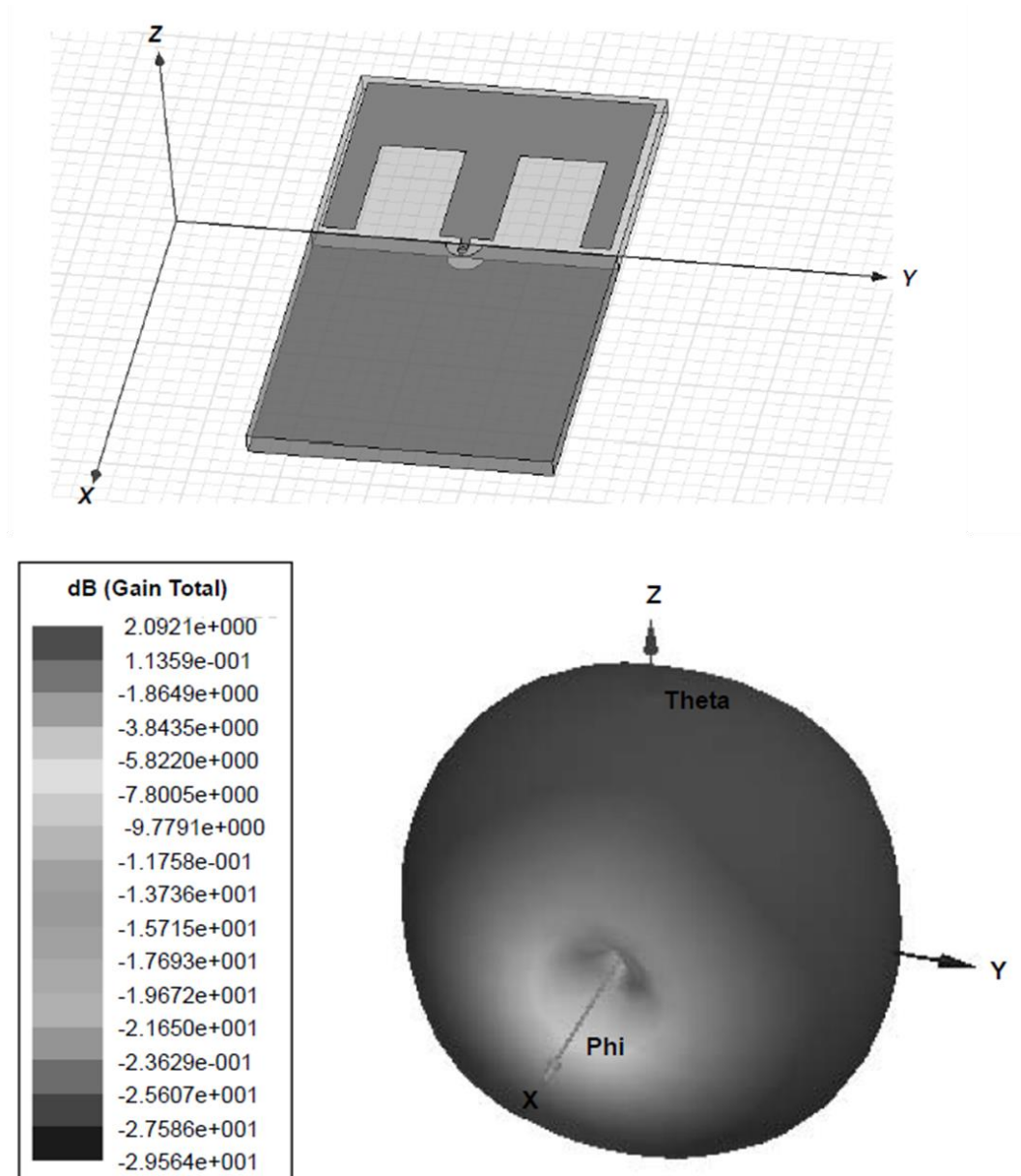
Gambar 2.13 Modul Microchip MRF24J40MA [22]

Bentuk dari modul radio ini adalah *dual chip*. Modul ini hanya berfungsi sebagai *transceiver* dan *protocol stack* disediakan dari kontroler luar yang bukan bagian dari *transceiver* ini. Salah satu bagian terpenting dari perancangan perangkat keras adalah pembuatan *board* untuk modul ini karena hal ini menyangkut kualitas konektivitas jaringan. Ada dua hal yang harus diperhatikan, yaitu bentuk *extension board* dan juga posisi antenna relatif terhadap penempatan alat. Gambar 2.14 menunjukkan petunjuk mendasar dari pembuatan *extension board* yang disarankan oleh *datasheet* MRF24J40MA.



Gambar 2.14 Petunjuk *mounting* modul MRF2J40MA [23]

Posisi dari antenna juga memengaruhi kualitas konektivitas. Gambar 2.17 menunjukkan pola radiasi dari antenna. Berdasarkan dari petunjuk ini, penulis merancang *extension board* untuk menyangga antenna dalam posisi vertikal.



Gambar 2.15 Gambar pola radiasi dalam bentuk 3D [23]

#### 2.6.4 Libcoap

Libcoap adalah library untuk protokol CoAP yang ditulis dalam bahasa C. Libcoap didesain untuk berjalan pada alat-alat *embedded* dan sistem komputer dengan sistem operasi berbasis POSIX. Proses pembuatan aplikasi dengan *libcoap* adalah dengan membuat *socket* dengan spesifikasi UDP (*socket datagram*) lalu payload akan dienkapsulasi dengan protokol CoAP menggunakan library *libcoap* sebelum dienkapsulasi oleh UDP.

### 2.6.5 Apache2 Web Server, URL Rewriting, dan CGI

Apache merupakan *web server* yang sangat populer di sistem berbasis Linux [24]. *Web server* digunakan untuk menangani permintaan dari *client* yang biasanya melakukan permintaan dari aplikasi *browsers* seperti Mozilla Firefox, Opera, Google Chrome, dan sebagainya. Apache *web server* dibuat dengan bahasa C dan XML [25].

Untuk dapat mengikuti standar penulisan URL sesuai dengan petunjuk IETF yang dibahas pada Subbab 2.3.2, diperlukan URL *rewrite*. Dengan URL *rewrite*, server dapat dikonfigurasi untuk menerima URL tertentu yang berbeda dengan *default* URL. Contohnya, jika server awalnya hanya menyediakan akses ke file *dalem.php* dengan URL `http://tes.com/dalem.php`, maka dengan URL *rewrite* bisa saja file tersebut diakses dengan cara `http://tes.com/luar/`. Untuk dapat melakukan URL *rewrite*, salah satu cara yang dapat dilakukan adalah dengan memberi aturan URL *rewrite* pada file *.htaccess* yang ada di *root* directory dari server. Karakter yang diterima pada aturan URL *rewrite* ditulis dengan aturan *regular expression*.

Salah satu fitur dari Apache2 adalah *Common Gateway Interface* (CGI). CGI adalah sebuah metode yang digunakan *web server* untuk menjalankan program yang akan dijalankan di *web* [26]. Program CGI dapat ditulis dalam bahasa apapun, tentu saja program tersebut harus dapat dijalankan di sistem operasi *web server*, umumnya dapat ditulis di Perl, C, Java, dan sebagainya. Dalam penelitian ini, penulis menggunakan C sebagai bahasa yang digunakan untuk membuat program CGI. Untuk membuat program CGI, hanya perlu meng-*compile* program dengan hasil berekstensi *.cgi*.

## BAB 3

### PERANCANGAN HTTP-COAP *PROXY*

Bab ini memberi paparan mengenai perancangan sistem yang akan dibangun. Dalam membangun sistem ini, tahapan-tahapan yang dijalani meliputi mengumpulkan *system requirements*, desain, implementasi, dan testing, seperti tahap pada *Software Development Life Cycle* (SDLC) [27]. Untuk memberi ilustrasi, metode yang digunakan adalah dengan *Unified Model Language* (UML).

#### 3.1 Tinjauan Perancangan *Border Router* yang Sudah Ada

Tinjauan penelitian lain yang sudah dilakukan bertujuan untuk mencari referensi dalam merancang HTTP-CoAP *proxy*. Berikut akan dibahas mengenai mekanisme perancangan dari empat penelitian yang telah dilakukan sebelumnya.

Lerche et al. [11] menerapkan *cross-protocol proxy* pada rancangan *border router* untuk membentuk paradigma *Web of Things*. Selain *cross-protocol proxy*, Lerche et al. juga menambahkan fitur *caching* untuk mereduksi *network traffic*. *Proxy* ini tidak hanya bekerja untuk IPv6, tetapi juga IPv4. Untuk *proxy*, sistem menggunakan *jCoAP library* yang ditulis dalam Java, untuk *caching*, Lerche et al. mengajukan adanya waktu pengiriman dalam CoAP *header* seperti pada *header* di HTTP. Lerche et al. mengimplementasikan TelosB yang berbasis Erbium sebagai CoAP server dan *Firefox plugin Copper* sebagai klien. Namun sebagai tambahan, sistem keamanan belum ditambahkan dalam *proxy* dan dijadikan pekerjaan yang bisa dilakukan di masa depan.

Castellani et al. dalam [28] melakukan dua pendekatan dalam membangun *proxy*, yaitu dengan *framework* bernama WebThings yang berbasis UNIX/Linux dan dengan pencegatan HTTP-CoAP menggunakan Squid yang ditambahkan ke modul *cross-protocol proxy*. Penelitian yang dilakukan Castellani et al. tidak menggunakan 6LoWPAN untuk *constrained networks*. Hasil dari penelitian ini adalah implementasi *proxy* yang *compatible* dengan arsitektur REST.

Jeong et al. dalam [29] mengimplementasikan HTTP-CoAP *proxy* dalam bentuk *reverse proxy* yang memanfaatkan CGI untuk melakukan pemetaan

HTTP-CoAP. Penelitian Jeong et al. berfokus pada implementasi pemetaan HTTP-CoAP namun tidak mengimplementasikan fitur *caching*. Hasil dari penelitian itu adalah diimplementasikannya sebuah HTTP-CoAP *proxy* yang tidak membutuhkan konfigurasi pada *client* sehingga diharapkan dapat menjadi standar *web* yang bersifat *de facto*.

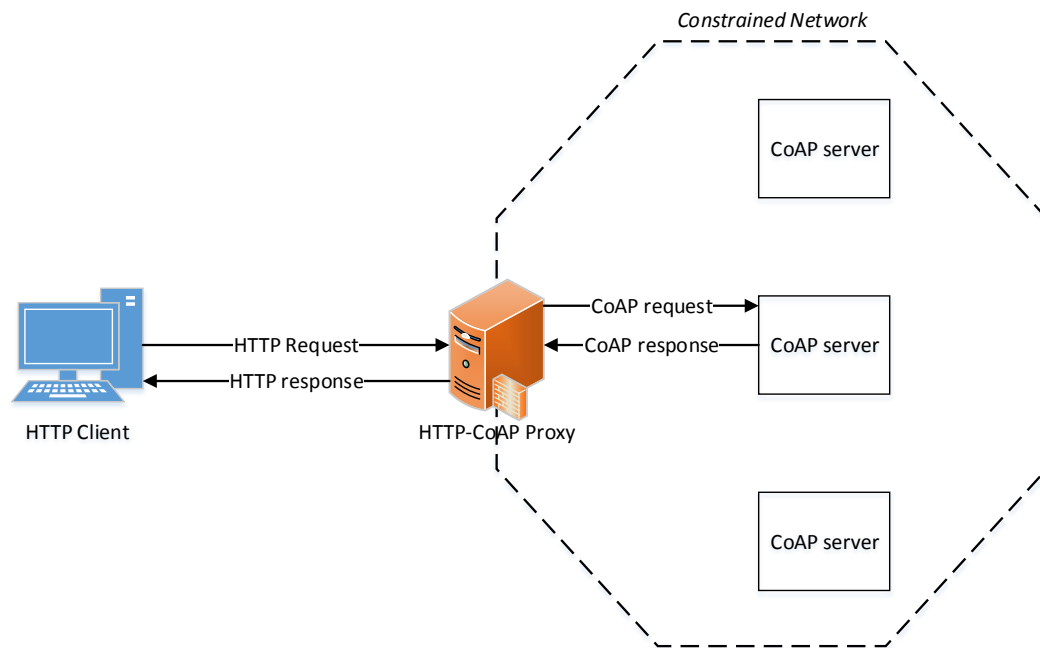
Penelitian yang dilakukan oleh Ludovici dan Calveras [30] meliputi perancangan dan pengimplementasian HTTP-CoAP *proxy* dan evaluasi pengaksesan dengan protokol HTTP dan *WebSocket*. *Proxy* pada penelitian tersebut terdiri dari beberapa modul, yaitu modul CoAP (untuk melakukan *request* ke CoAP sensor), modul *web server*, *cache*, dan *resource directory*. Pada penelitian tersebut digunakan dua teknik untuk mengumpulkan data dari sensor, yaitu dengan *WebSocket* dan HTTP *long-polling*. Evaluasi yang diuji terhadap perancangan *proxy* adalah dengan mengukur *latency* dari penggunaan request *WebSocket* dan HTTP *long-polling*. Hasil dari penelitian tersebut menunjukkan *WebSocket* cocok digunakan untuk aplikasi yang bersifat *long-lived* (dengan protokol *observe* dimana *client* akan terus diberikan data baru jika sensor memiliki data yang juga baru), sedangkan HTTP *long-polling* cocok untuk aplikasi yang bersifat *short-lived* (data akan diberikan jika *client* meminta), kedua kesimpulan tersebut diambil atas alasan kecilnya nilai *latency*.

### 3.2 Requirement Analysis (Pertimbangan Perancangan)

Pertimbangan utama dalam merancang *proxy* adalah petunjuk pemetaan protokol HTTP ke CoAP [14] dan petunjuk representasi *payload* dari CoAP [31] yang masih berstatus draft IETF (belum standar RFC hingga penelitian ini dikerjakan). Pertimbangan lainnya dianalisa sebagai kebutuhan umum sistem dan akan dibahas pada Subbab 3.2.

#### 3.2.1 Skenario Fungsionalitas Sistem

Desain dari skenario sistem dapat dilihat pada Gambar 3.1 yang menunjukkan tiga komponen, yaitu HTTP client (dapat berupa komputer dengan *web browser*), HTTP-CoAP *reverse proxy*, dan CoAP server yang merupakan sensor node yang ada di *constrained network*.



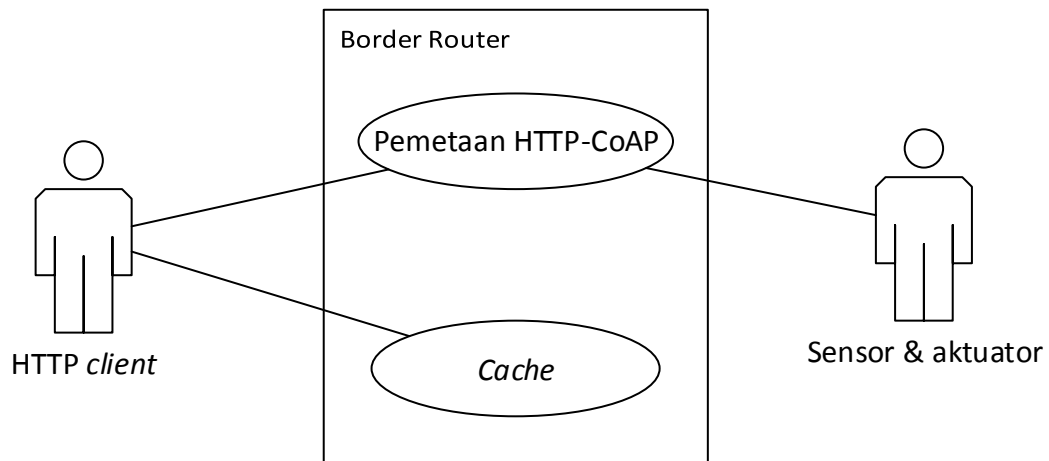
Gambar 3.1 Skenario *Deployment* Perancangan

Pengguna dari sistem ini diharapkan dapat mengakses data dari jaringan sensor nirkabel dari perangkat komputer yang terhubung ke internet. Dari survey yang telah dilakukan Kovatsch dalam [32], 77% pengguna lebih memilih interaksi dengan halaman *web*. Kovatsch mengajukan suatu metode untuk mengakses server dengan protokol CoAP dengan *add-on* Firefox bernama Copper. Sayangnya, walaupun Copper ditulis dalam JavaScript, CoAP *request* tidak dapat dilakukan dari skrip lain yang dibuat *custom*.

Penulis mengumpulkan hasil pengumpulan *requirements* dan berikut adalah *requirements* tersebut:

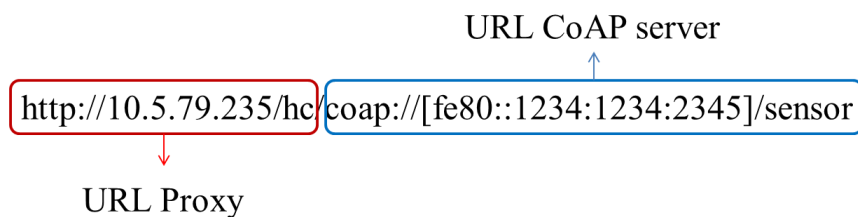
- Pengguna dapat mengakses data dengan metode GET pada URL sebagai parameter, dan data akan dikembalikan dalam bentuk JSON.
- *Proxy* harus dapat diakses beberapa *client* secara bersamaan dan tetap memberikan respon yang sesuai.

Fungsi diatas adalah fungsi dasar yang ada pada sistem atas memenuhi kebutuhan pengguna. Gambar 3.2 menggambarkan sistem dalam bentuk diagram *use case*.



Gambar 3.2 Diagram *Use Case* dari Proxy

Proses pemetaan protokol HTTP ke CoAP mengikuti *draft* dari IETF yang menjelaskan mengenai pemetaan HTTP ke CoAP pada *reverse proxy* [14]. Dalam skenarionya, anggap *proxy* memiliki alamat IPv4 10.5.79.235 dan *proxy* ini dapat diakses oleh HTTP client dengan `http://10.5.79.235/hc` (directory `hc` merupakan singkatan dari HTTP-CoAP). Saat suatu client ingin mendapatkan akses ke sensor node dengan alamat IPv6 `fe80::1234:1234:2345` dalam format CoAP maka client menggunakan URL `http://10.5.79.235/hc/coap://[fe80::1234:1234:2345]/`, sesuai dengan [14]. Gambar 3.3 menggambarkan mekanisme pengaksesan CoAP server dari klien HTTP ini.



Gambar 3.3 Mekanisme URL Untuk CoAP Server

Data yang akan didapatkan pengguna berbentuk JSON. Misalnya, jika pengguna melakukan permintaan data sensor dengan URL `http://10.5.79.235/hc/coap://[fe80::1234:1234:2345]/sensor`, maka hasilnya adalah `{"sensor":80}` yang berarti data sensor bernilai



80. Pengembalian data untuk *resource discovery* juga dalam bentuk JSON yang mengikuti *draft* standar IETF [31] seperti yang terlihat pada Gambar 3.4.

```
</sensors>;ct=40;title="Sensor Index",
</sensors/temp>;rt="temperature-c";if="sensor",
</sensors/light>;rt="light-lux";if="sensor",
<http://www.example.com/sensors/t123>;anchor="/sensors/temp"
;rel="describedby",
</t>;anchor="/sensors/temp";rel="alternate"
```

Gambar 3.4 Respon CoAP *Resource Discovery*

Respon CoAP pada Gambar 3.4 diubah menjadi bentuk JSON seperti pada Gambar 3.5.

```
"[{\"href\":\"/sensors\",\"ct\":\"40\",\"title\":\"Sensor
  Index\"},{\"href\":\"/sensors/temp\",\"rt\":\"temperature-
c\",\"if\":\"sensor\"},{\"href\":\"/sensors/light\",\"rt\":\"lig
ht-
lux\",\"if\":\"sensor\"},{\"href\":\"http://www.example.com/
sensors/
  t123\",\"anchor\":\"/sensors/
temp\",\"rel\":\"describedby\"},{\"href\":\"/t\",\"anchor\":\"/s
ensors/
  temp\",\"rel\":\"alternate\"}] "
```

Gambar 3.5 Bentuk JSON dari CoAP *Resource Discovery*

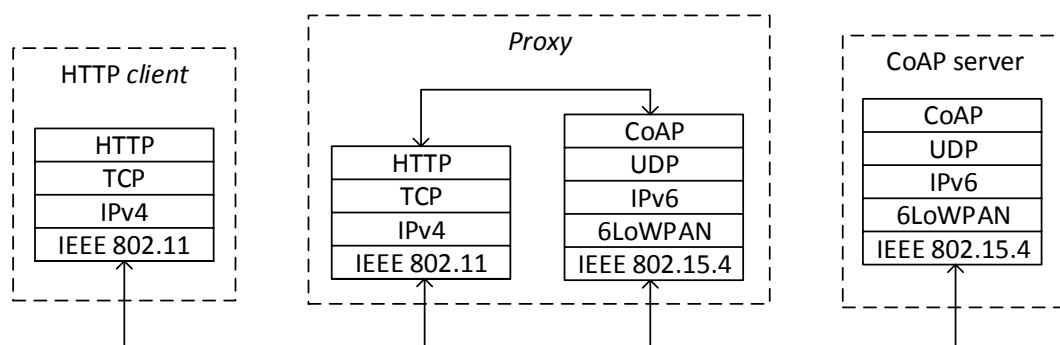
Salah satu hal yang menjadi pertimbangan dalam perancangan *proxy* ini adalah fitur CoAP *multicast*. Sesuai dengan panduan pemetaan HTTP-CoAP pada *reverse proxy* [14], sebuah HTTP-CoAP *proxy* dibolehkan untuk mengijinkan permintaan *multicast*, namun jika tidak mengijinkan, seharusnya respon 403 Forbidden diberikan ke HTTP *client*. CoAP *multicast* dapat memberatkan *network traffic* pada *constrained network*, membuka kemungkinan untuk diserang dengan *Denial-of-Service* (DoS), dan dari perspektif privasi, *multicast* terlalu mengekspos *resource* yang ada di jaringan. Pada penelitian skripsi ini, fitur CoAP *multicast* tidak diajukan ke dalam sistem karena alasan keamanan yang di luar dari batasan masalah.

### 3.2.2 Fitur/*Resource* yang dimiliki CoAP Server

CoAP server memiliki tiga *resource* yang dapat diakses, yang pertama adalah sensor *flowmeter*, input ADC yang diemulasikan dengan potensiometer, yang ketiga adalah GPIO yang dapat diatur untuk menjalankan aktuator. CoAP server memberikan umur data yang dianggap valid untuk disimpan di *cache* yang terletak pada *proxy* dalam bentuk CoAP header.

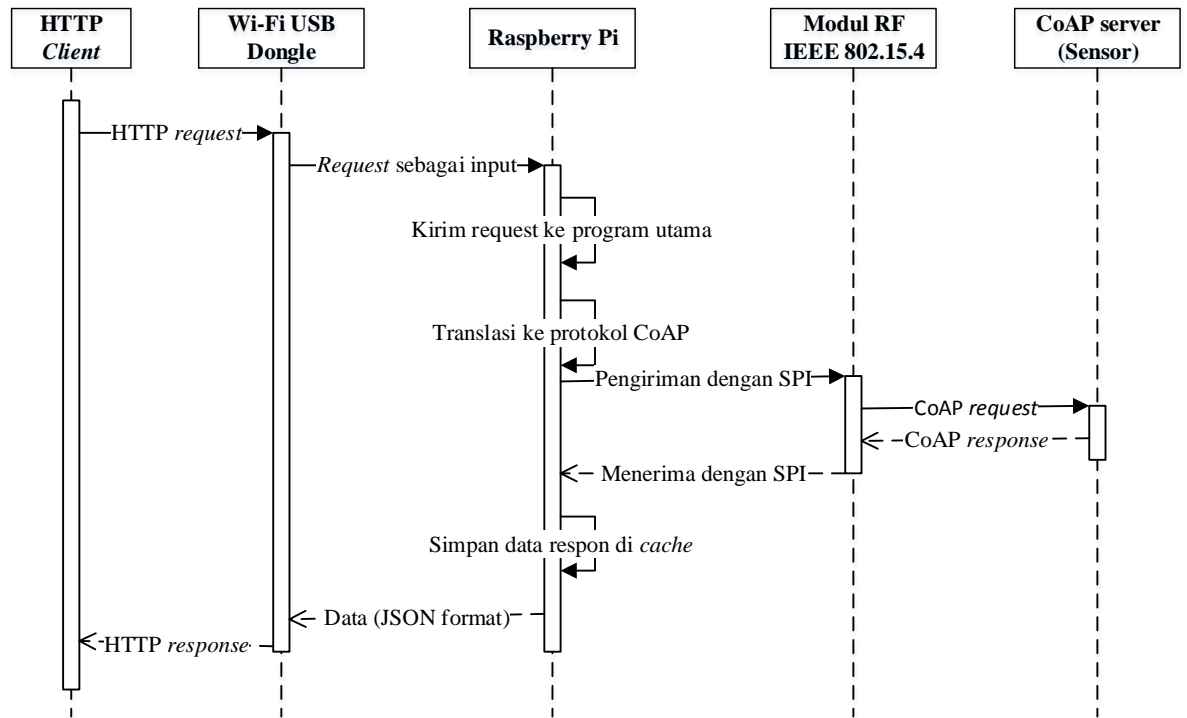
### 3.3 Perancangan Sistem dan Perangkat Lunak

Pada subbab ini akan dijelaskan mengenai perancangan sistem dan juga perangkat lunak dari HTTP-CoAP *proxy*. *Proxy* didesain dengan skenario penggunaan sesuai dengan Gambar 3.1, yang menunjukkan *proxy* berguna sebagai 6LoWPAN *border router* dan juga sebagai *gateway* dari jaringan CoAP. Gambar 3.6 menunjukkan arsitektur *proxy* dari sudut pandang *protocol stack*.



Gambar 3.6 *Protocol Stack* dari Sistem

Secara urutan dari proses berjalannya sistem yang ditunjukkan Gambar 3.6, Gambar 3.7 merepresentasikan urutan kerja sistem dalam bentuk *sequence diagram*.

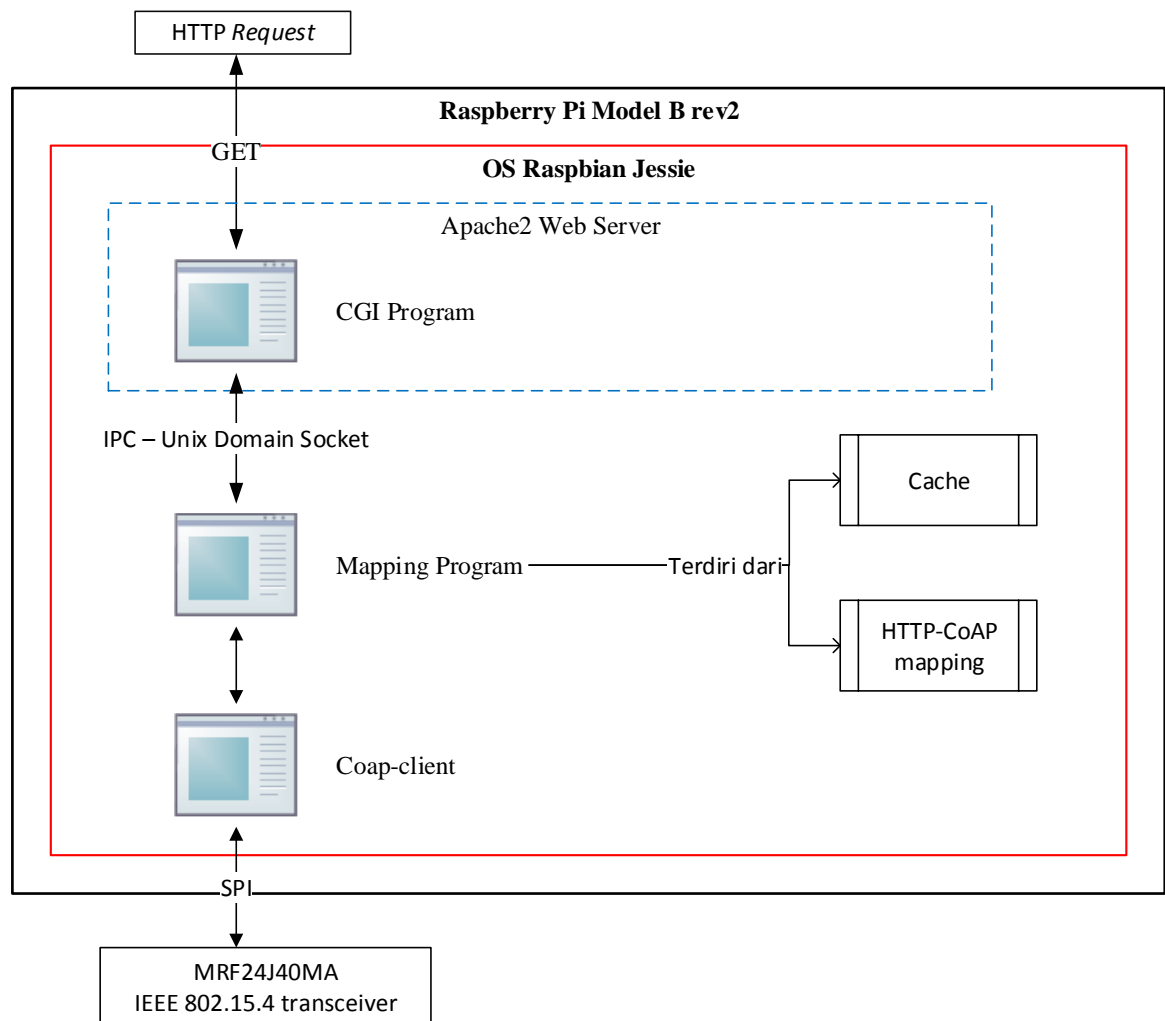


Gambar 3.7 Diagram *Sequence* dari Proxy

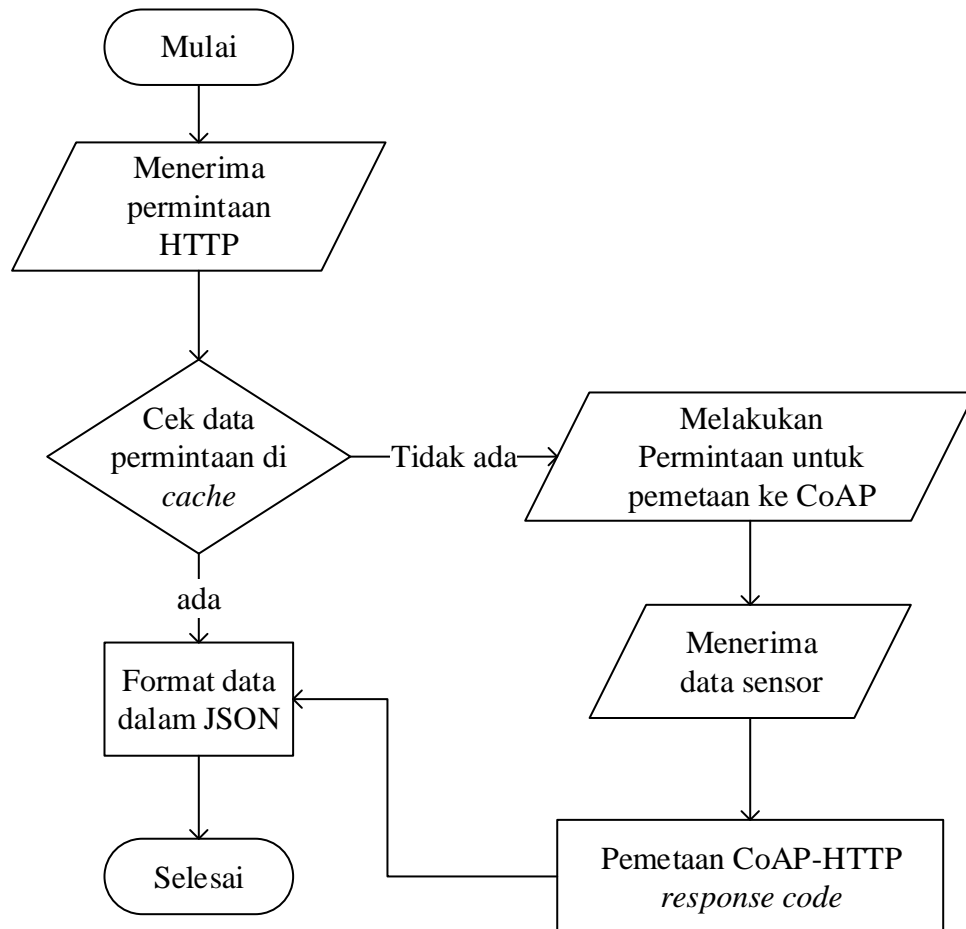
Dalam mewujudkan sistem *proxy*, perangkat lunak dari *proxy* ini dirancang untuk memiliki empat bagian:

- Modul *web* server
- Modul pemetaan HTTP-CoAP
- Modul *caching*
- Modul CoAP

Modul-modul tersebut dapat digambarkan dalam blok diagram pada Gambar 3.8.

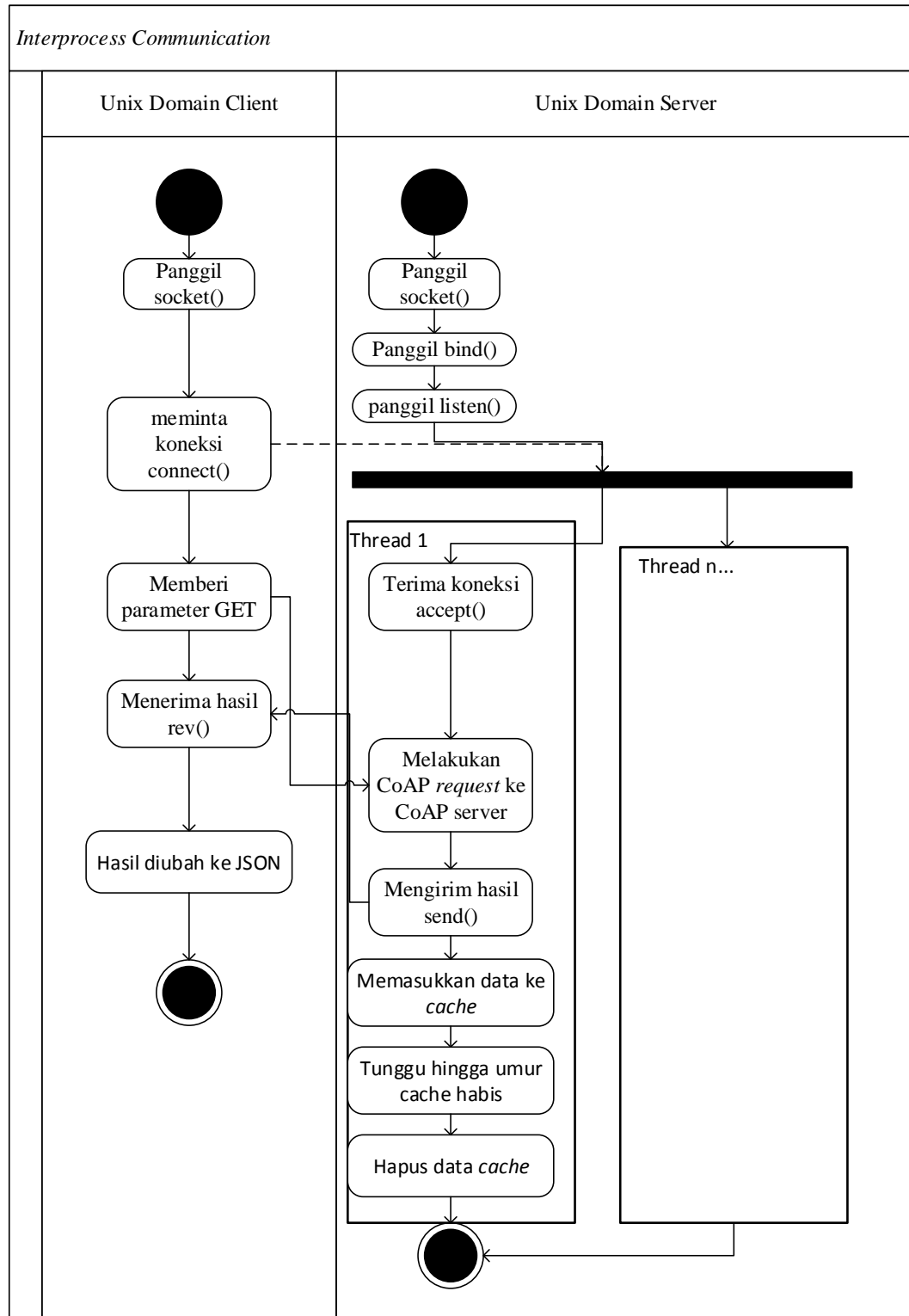
Gambar 3.8 Diagram Blok Sistem *Proxy*

Saat modul Apache *web* server menerima HTTP *request* dari client, maka apache akan menjalankan program CGI yang ditulis dalam bahasa C. Program CGI ini berfungsi untuk mengambil *query string* dari HTTP *request* yang mana merupakan URL untuk mengakses CoAP server dan mengembalikan data dari sensor dalam format JSON. Setiap ada permintaan HTTP baru, program CGI akan dijalankan sebagai satu proses per permintaan, sehingga diperlukannya suatu mekanisme penanganan akses ke transceiver yang akan beroperasi ke jaringan sensor. Program CGI ini akan melakukan *request* ke modul program pemetaan HTTP-CoAP menggunakan UNIX Domain *Socket*. Setelah mendapatkan data, program CGI ini juga akan mengubah format *response* dari CoAP menjadi format *string* JSON. Program ini juga bertanggung jawab memetakan *response code* CoAP ke HTTP. Gambar 3.9 menggambarkan algoritma dari program CGI ini.



Gambar 3.9 Flowchart Program CGI

Di program pemetaan HTTP-CoAP, tiap permintaan dari program CGI akan dicek ke modul *cache*. Jika ada di modul *cache* permintaan yang sesuai, maka data akan diambil dari *cache* dan tidak akan mengambil langsung ke sensor sebagai salah satu mekanisme *congestion control*. Jika tidak ada, maka permintaan akan diteruskan ke CoAP server dan hasil dari permintaan tersebut akan dimasukkan ke *cache*. Tiap permintaan dari program CGI akan dilayani dengan satu *thread* agar *proxy* dapat diakses secara bersamaan (*concurrent*). Agar menghindari tidak konsistennya data yang dihasilkan, proses mengakses *cache* dan CoAP server dianggap sebagai sebuah *critical section* yang harus dikunci menggunakan *mutual exclusion (mutex)*. Gambar 3.10 menggambarkan interaksi CGI program dengan modul pemetaan HTTP-CoAP.



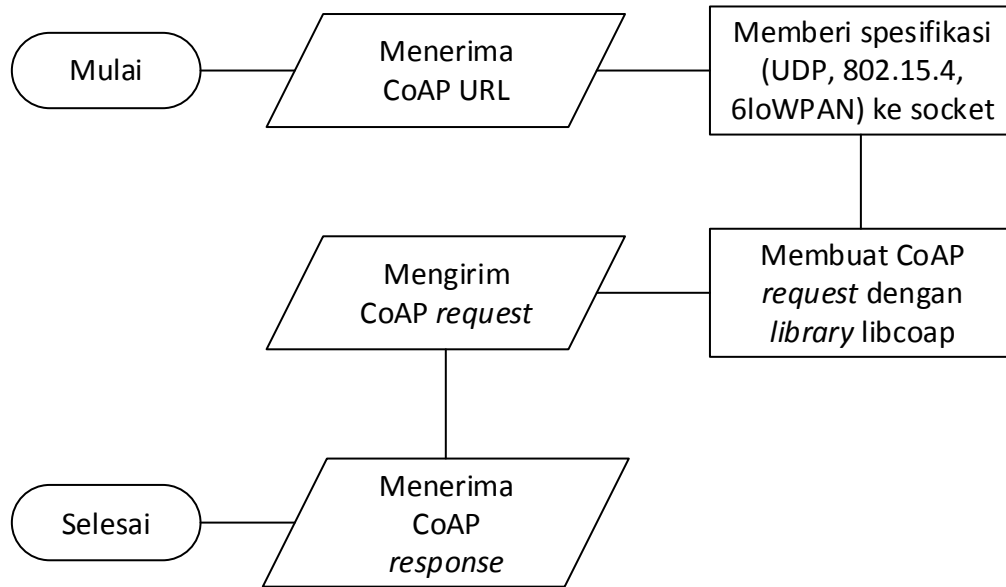
Gambar 3.10 Diagram Aktivitas Program CGI dan Pemetaan HTTP-CoAP

Modul *cache* berfungsi sebagai modul penyimpan data permintaan dari *client* yang mencegah terlalu ramainya *traffic* di jaringan sensor. Valid atau

tidaknya data yang ada di *cache* ditentukan oleh CoAP server yang memberi umur data di CoAP *header*. Data yang disimpan adalah URL *request*, hasil data sensor, umur data, dan *timestamp*. Data-data tersebut direpresentasikan dalam sebuah *linked list* dan akan dihapus entitasnya jika waktunya sudah habis. *Cache* yang dirancang merupakan *cache* yang sederhana tanpa mekanisme *replacement* jika memori dari *cache* penuh. Hal ini dilakukan karena *cache* ukurannya kecil dan disimpan di *main memory*. Besar dari *cache* bersifat deterministik, dan dapat dihitung dengan mengalikan jumlah CoAP server dikalikan dengan jumlah *resource* pada tiap server dikalikan dengan ukuran tiap objek di *cache* seperti yang dirumuskan pada Persamaan 3.1.

$$Max\ cache = \left( \sum (CoAP\ server) * (resource) \right) * size\ per\ objek \quad (3.1)$$

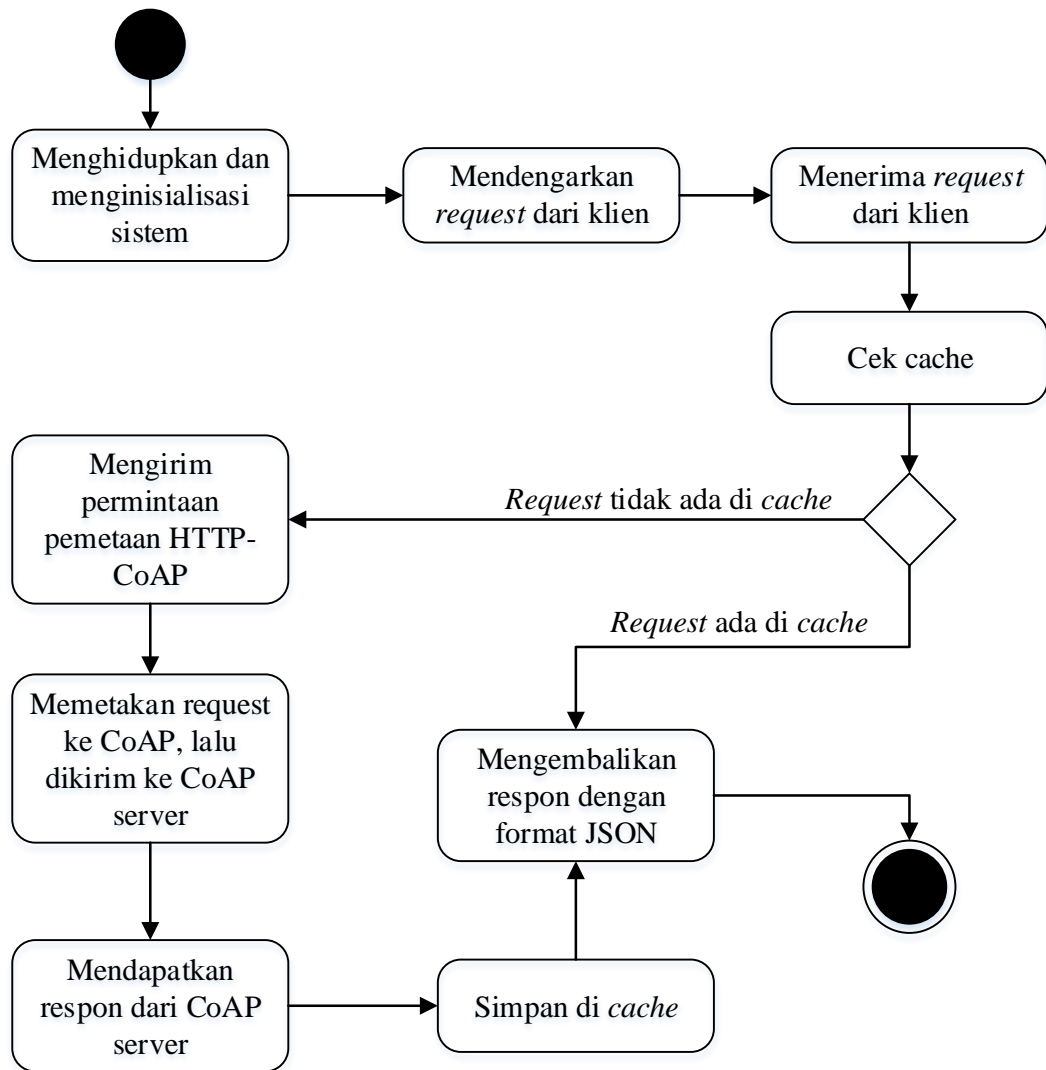
Modul CoAP berfungsi untuk melakukan *request* ke CoAP server dan menerima kembali hasil dari server. Modul ini menggunakan *library* Libcoap untuk enkapsulasi *payload* dan format *request* pada lapisan aplikasi dengan menambahkan *header* CoAP. Modul ini juga menspesifikasi UDP untuk *transport layer*. Modul ini dijalankan oleh modul pemetaan HTTP-CoAP dan datanya pun akan digunakan oleh modul pemetaan HTTP-CoAP. Gambar 3.11 menggambarkan kerja modul ini dalam bentuk *flowchart*.



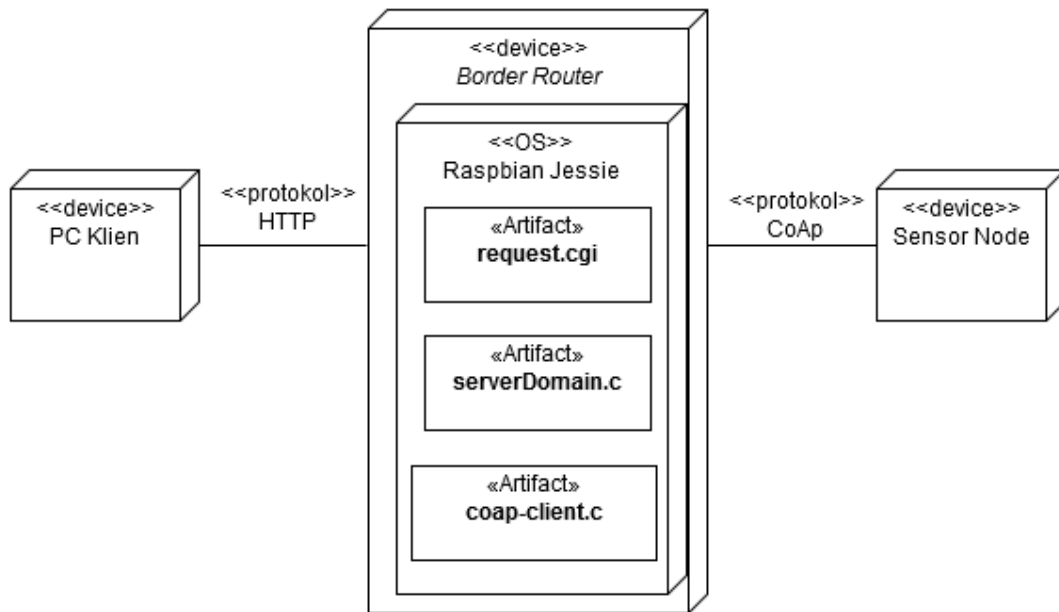
Gambar 3.11 *Flowchart* dari Modul CoAP Client

Jika modul-modul program yang sudah dijelaskan sebelumnya digabung, akan membentuk suatu aktifitas yang diilustrasikan oleh UML *activity diagram* yang terlihat pada Gambar 3.12.



Gambar 3.12 Diagram Aktivitas *Proxy*

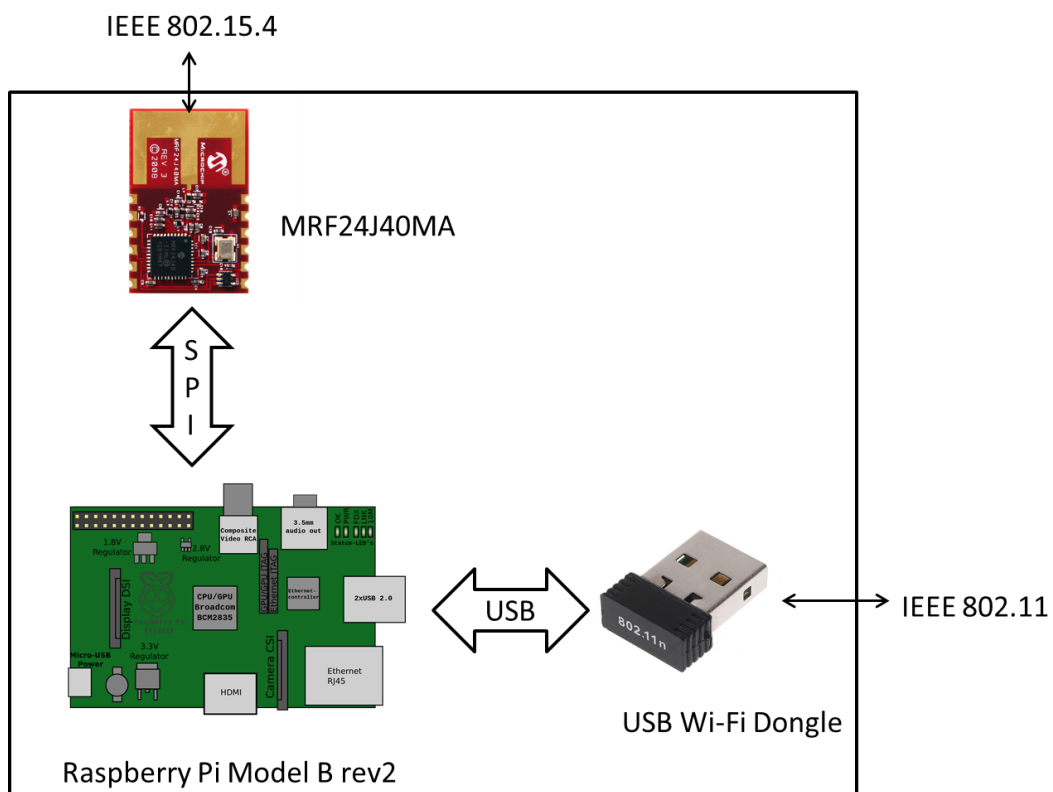
Untuk menggambarkan interaksi perangkat keras dan perangkat lunak dari sistem, dan juga bagian detail lainnya dari perancangan sistem, *deployment diagram* pada Gambar 3.13 digunakan. Semua program ditulis dalam bahasa C dan melibatkan UNIX *socket* dan *thread programming*.



Gambar 3.13 Diagram *Deployment* dari *Proxy*

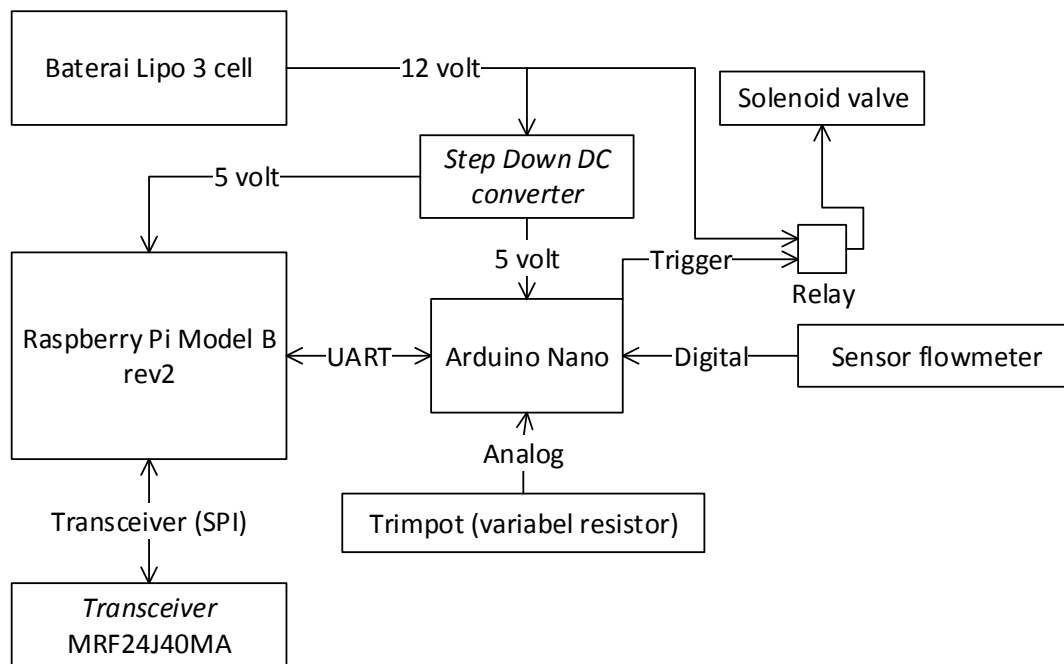
### 3.4 Perancangan Perangkat Keras

Sitem perangkat keras dibagi menjadi dua alat, yaitu HTTP-CoAP *proxy* dan CoAP server. HTTP-CoAP *proxy* terdiri dari Raspberry Pi Model B rev2, transceiver IEEE 802.15.4 MRF24J40, dan USB Wifi Dongle. Gambar 3.14 menggambarkan rancangan perangkat keras untuk HTTP-CoAP *proxy*.



Gambar 3.14 Rancangan Perangkat Keras *Proxy*

Untuk bagian CoAP server, alat ini terdiri dari Raspberry Pi Model B rev2, Arduino Nano, variabel resistor, sensor *flowmeter*, dan solenoid valve 12 volt. Tujuan digunakannya Arduino Nano dalam sistem ini adalah karena Raspberry Pi tidak memiliki ADC di dalamnya, sehingga untuk memodelkan sistem sensor ADC diperlukan Arduino yang hasil pembacaan ADC-nya akan dikirim secara serial ke Raspberry Pi. Gambar 3.15 menggambarkan rancangan perangkat keras dari CoAP server.

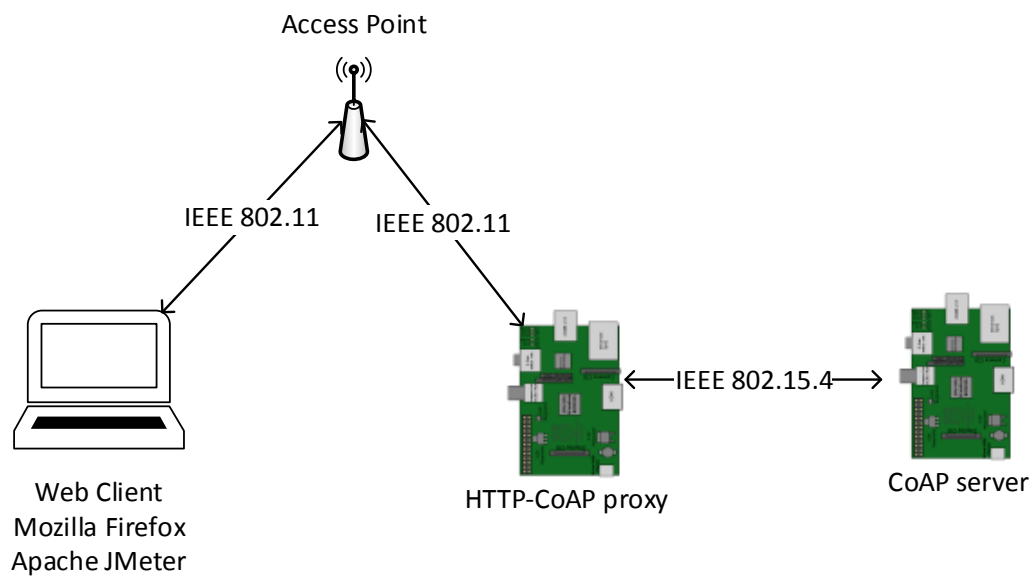


Gambar 3.15 Rancangan Perangkat Keras CoAP Server

### 3.5 Skenario Pengujian dan Pengukuran Performa HTTP-CoAP Proxy

Eksperimen pengujian dan pengukuran dilakukan dengan metode HTTP yang bersifat *long lived*, dimana sebuah TCP yang sama digunakan untuk mengirim dan menerima permintaan dan respon HTTP. Untuk pegujian, penulis mengimplementasikan sebuah skenario jaringan sensor nirkabel yang bersifat single-hop seperti yang terlihat pada Gambar 3.16 dengan besar paket yang ditransmisikan sebesar 228 bytes dengan umur data untuk di *cache* selama 30 detik. Evaluasi performa akan diuji dengan menggunakan perangkat lunak bernama Apache JMeter. Apache JMeter dapat menguji *latency* dan *throughput* dengan mengemulasikan beberapa *concurrent client* yang mengakses *proxy* secara bersamaan.

Skenario kedua adalah untuk menguji efektifitas penggunaan cache dalam *proxy*. Untuk menguji hal ini, dibuatlah suatu skenario dimana *proxy* diberi permintaan dari HTTP *client* sebanyak 100 kali dalam satu menit dengan jeda antar *request* seragam. Variabel yang dikontrol dalam pengujian ini adalah umur data valid dalam *proxy* yang diberikan oleh CoAP server.



Gambar 3.16 Skenario Pengujian Sistem

Secara rinci, perangkat dari *web client* adalah sebagai berikut:

- Komputer laptop Asus A46
- Sistem operasi: Linux Ubuntu 14.04 LTS
- Prosesor: Intel® Core™ i5-3317U CPU @ 1.70 GHz
- Memori: 4.00 GB
- *Tools* untuk pengoperasian: Mozilla Firefox, Apache JMeter

Secara rinci, *access point* yang digunakan adalah:

- *Handphone* Samsung Galaxy S4
- Sistem Operasi: Android 5.0.1
- Prosesor: ARM Cortex-A15 1.2 GHz quad-core
- Memori: 2.00 GB

## BAB 4

### IMPLEMENTASI DAN EVALUASI

Bab ini membahas hasil implementasi dari sistem beserta evaluasi performa dari *proxy* yang dirancang dengan metode *load testing* dan *stress testing*. Analisa dilakukan berdasarkan *latency*, *throughput*, *error rate*, dan efektifitas *cache* yang dihasilkan dari testing.

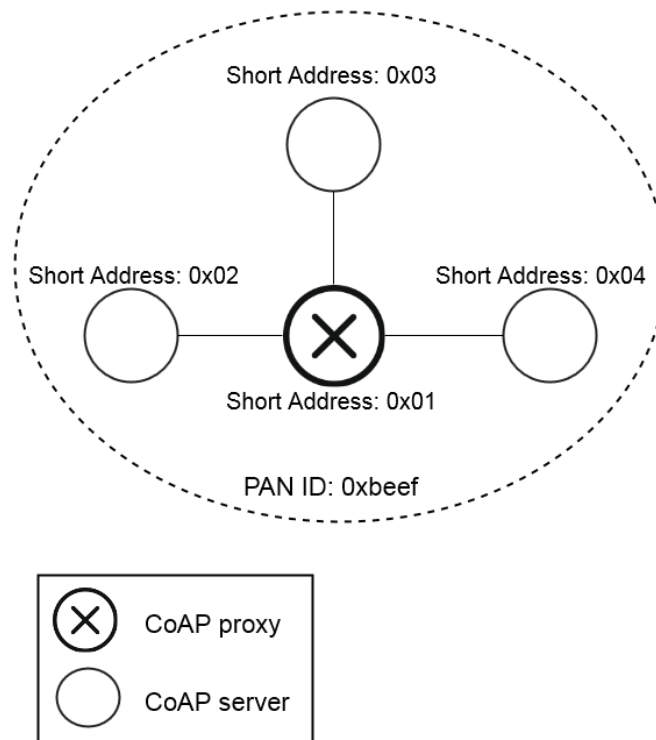
#### 4.1. Implementasi Sistem

Subbab ini menjelaskan detail penerapan rancangan yang dijelaskan pada Bab 3. Tahap implementasi dibagi menjadi dua bagian, yaitu implementasi perangkat keras dan implementasi perangkat lunak.

##### 4.1.1 Konfigurasi Raspberry Pi untuk 6LoWPAN Interface

6LoWPAN di Linux dikembangkan secara langsung dalam *kernel* Bluetooth-next dan sudah ada di *Kernel* resmi Linux sejak *kernel* versi 3.2 [17]. Untuk mendapatkan kondisi terbaru dari penerapan 6LoWPAN, dibutuhkan Bluetooth-next *kernel* dengan melakukan *cross-compile* dan memasukkan hasil kompilasi ke dalam Linux Raspbian yang ada di Raspberry Pi. Pada saat proses *cross-compile*, sangatlah penting untuk memasukkan driver MRF24J40, 6LoWPAN *interface*, dan modul IEEE802.15.4 ke dalam *kernel* agar modul-modul tersebut bisa digunakan.

Setelah berhasil dilakukan *cross-compile*, diperlukan program untuk mengkonfigurasi netlink 802.15.4 yang ada di Linux. Program yang digunakan untuk konfigurasi tersebut adalah *wpan-tools*. Dengan *wpan-tools*, pengguna dapat mengeset beberapa hal terkait dengan netlink 802.15.4, namun pada penelitian kali ini yang paling penting adalah mengatur ID dari personal area network dan juga *short address*. Pada penelitian ini PAN ID dan *short address* yang diset adalah 0xbeef dan 0x01 untuk CoAP server, dan juga 0x002, 0x003, dan 0x004 untuk tiap *short address* CoAP server. Gambar 4.1 memberi visualisasi dari pengaturan jaringan IEEE 802.15.4.



Gambar 4.1 Pengaturan Jaringan IEEE 802.15.4

Alamat IPv6 dari CoAP *proxy* dan CoAP server didapat dari IPv6 *stateless address configuration*, yang memanfaatkan MAC address dari *transceiver* dan EUI-64. Alamat IPv6, yang bersifat *link-local*, digunakan melalui *interface* yang bersesuaian digunakan untuk komunikasi dengan mekanisme 6LoWPAN. Gambar 4.2 menunjukkan fungsionalitas 6LoWPAN dengan ping6 ke alat lain dengan *interface* 6LoWPAN.

```

pi@raspberrypi:~ $ ping6 fe80::645b:6a78:23:de59%lowpan0
PING fe80::645b:6a78:23:de59%lowpan0(fe80::645b:6a78:23:de59) 56 data bytes
64 bytes from fe80::645b:6a78:23:de59: icmp_seq=1 ttl=64 time=24.2 ms
64 bytes from fe80::645b:6a78:23:de59: icmp_seq=2 ttl=64 time=13.2 ms
64 bytes from fe80::645b:6a78:23:de59: icmp_seq=3 ttl=64 time=13.6 ms
64 bytes from fe80::645b:6a78:23:de59: icmp_seq=4 ttl=64 time=11.0 ms
64 bytes from fe80::645b:6a78:23:de59: icmp_seq=5 ttl=64 time=13.0 ms
64 bytes from fe80::645b:6a78:23:de59: icmp_seq=6 ttl=64 time=17.7 ms
64 bytes from fe80::645b:6a78:23:de59: icmp_seq=7 ttl=64 time=13.6 ms
64 bytes from fe80::645b:6a78:23:de59: icmp_seq=8 ttl=64 time=12.3 ms
64 bytes from fe80::645b:6a78:23:de59: icmp_seq=9 ttl=64 time=11.2 ms

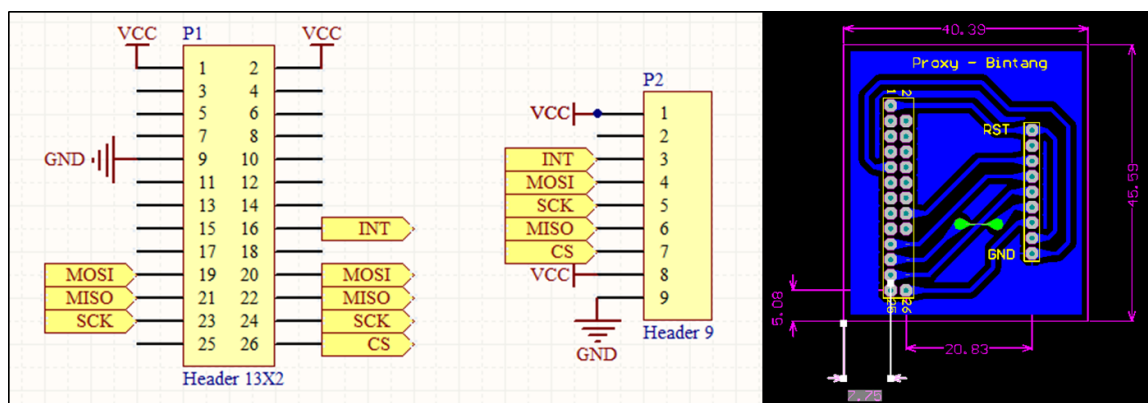
```

Gambar 4.2 Ping dengan 6LoWPAN

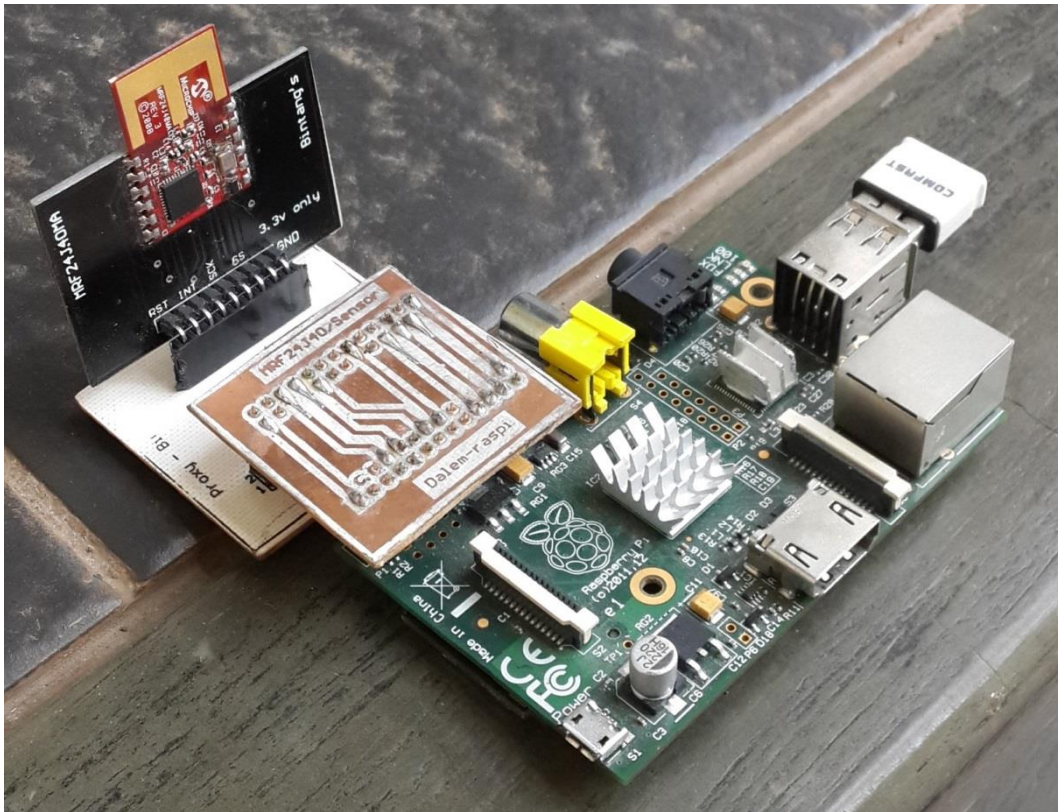
#### 4.1.2 Implementasi Perangkat Keras

Perangkat keras berupa rangkaian elektronik didesain dengan perangkat lunak bernama Altium Designer Summer 2009. Dua perangkat dibuat dengan bantuan perangkat lunak ini, yaitu rangkaian elektronik untuk HTTP-CoAP *proxy* dan CoAP server.

Untuk HTTP-CoAP *proxy*, rangkaian elektronik yang dirancang hanya berfungsi sebagai penyambung antara Raspberry Pi dengan *transceiver* MRF24J40MA. Gambar 4.3 menggambarkan skematik dan desain PCB dari *proxy* ini dan Gambar 4.4 menggambarkan bentuk fisik dari *proxy*.

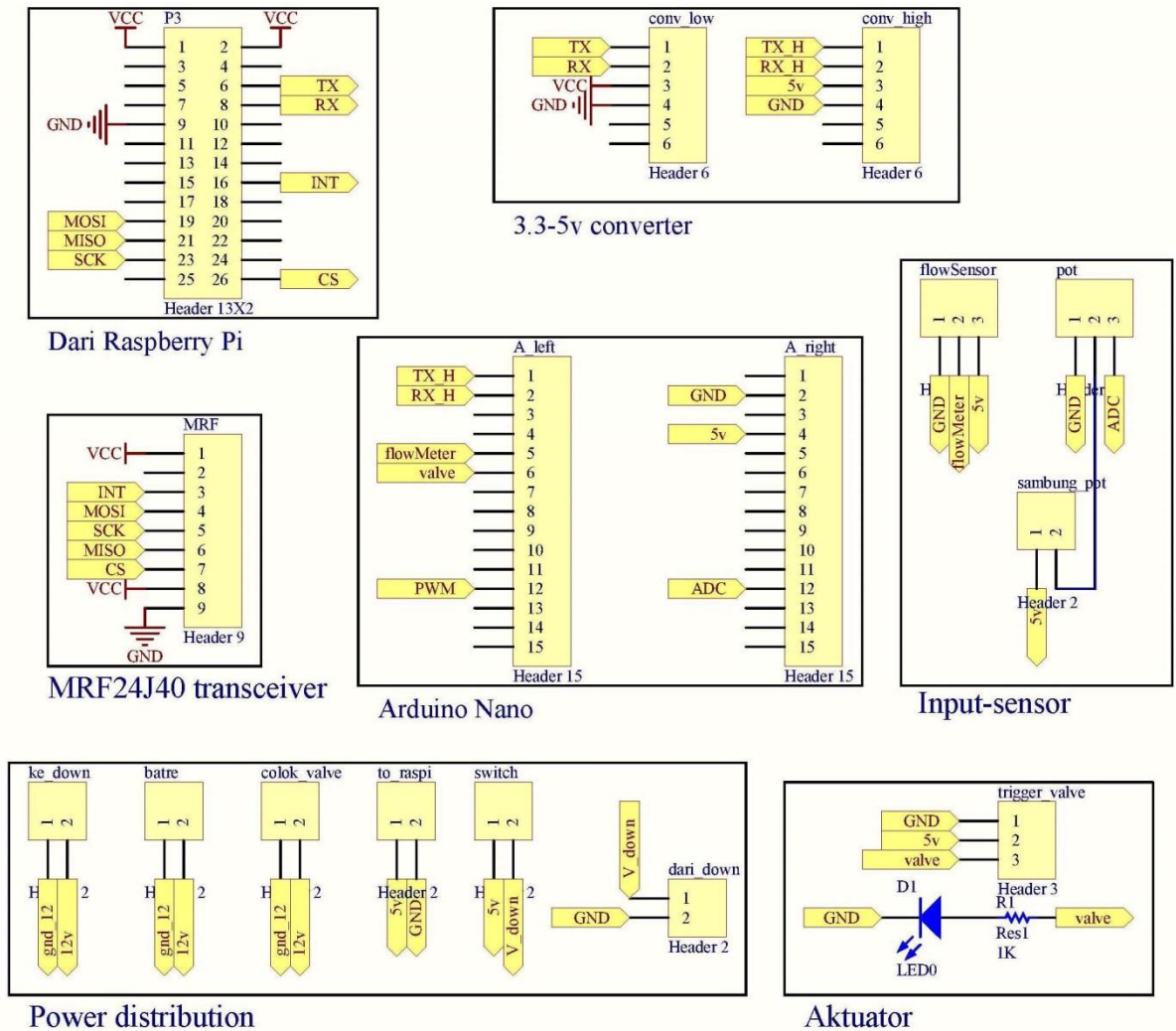
Gambar 4.3 Skematik dan Desain PCB *Proxy*



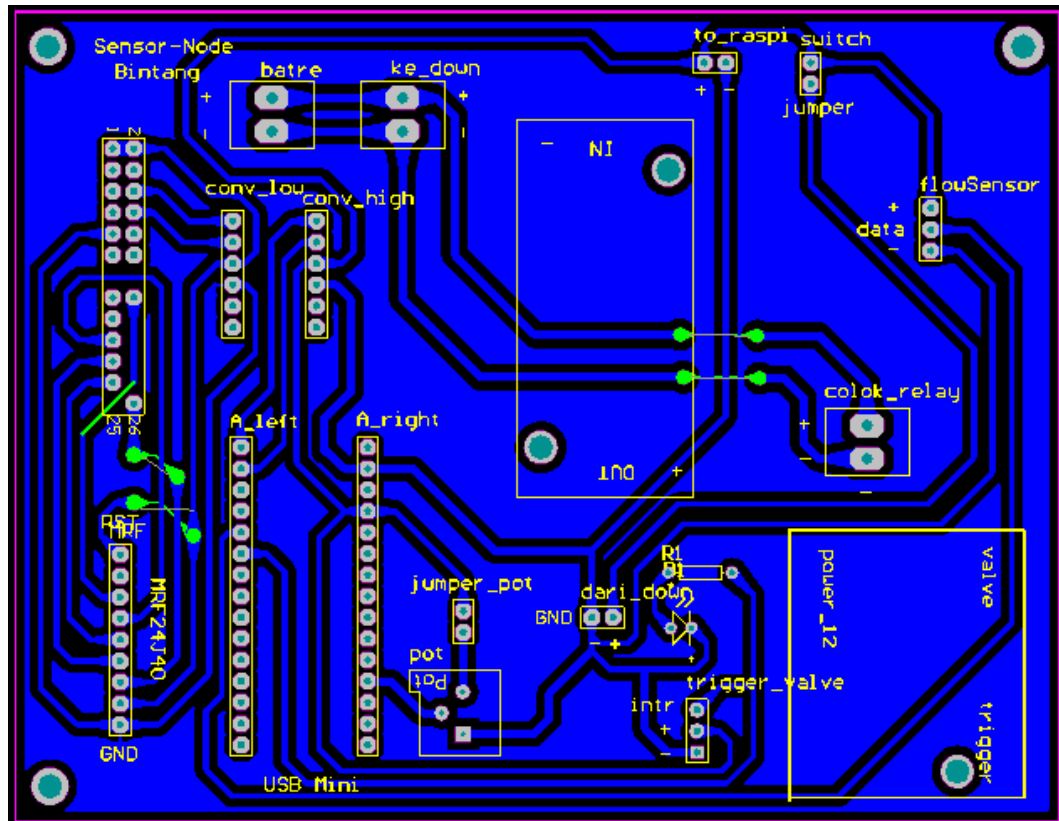


Gambar 4.4 Hasil Implementasi Perangkat Keras *Proxy*

Untuk bagian CoAP server, rangkaian elektronik yang dirancang berfungsi sebagai penghubung antar sensor-aktuator dengan mikrokontroler Arduino dan Raspberry Pi. Gambar 4.5 dan Gambar 4.6 menggambarkan hasil perancangan skematik dan PCB dan CoAP server.

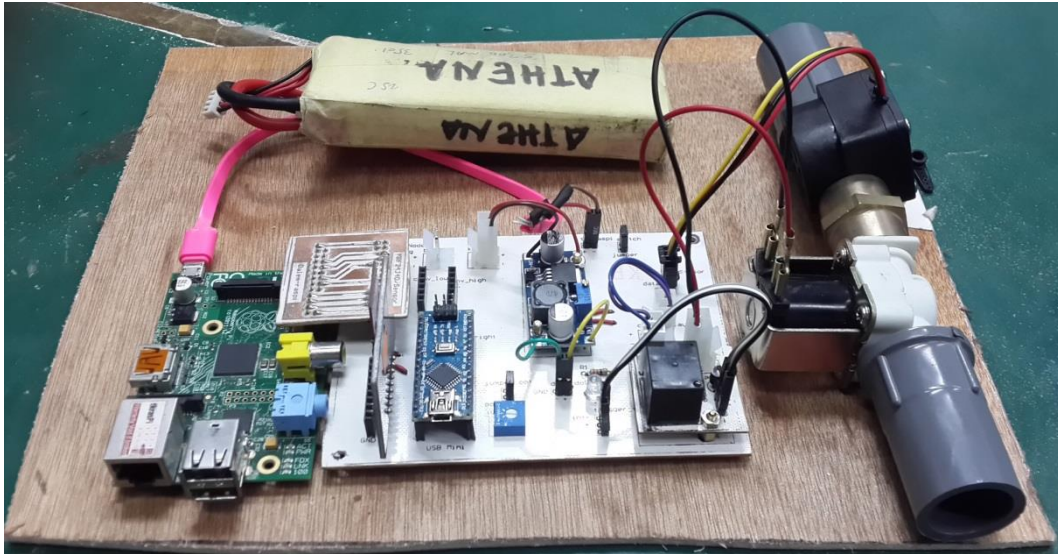


Gambar 4.5 Skematik CoAP Server



Gambar 4.6 Desain PCB CoAP Server

Hasil desain PCB diproduksi dengan cara memotong papan PCB, lalu mencetak hasil desain ke papan PCB dengan cara menyetrikan kertas film/*art paper* ke papan PCB, lalu di etsa dan akhirnya dibor. Tahap berikutnya adalah menyolder komponen ke papan PCB. Hasil implementasi ke bentuk fisik dari CoAP server dapat dilihat pada Gambar 4.7.



Gambar 4.7 Hasil Implementasi Perangkat Keras CoAP Server

#### 4.1.3 Implementasi Perangkat Lunak

Perangkat lunak pada sistem ini, baik pada HTTP-CoAP *proxy* maupun pada CoAP server dibuat menggunakan bahasa C. Pada HTTP-CoAP *proxy*, secara garis besar terdiri dari tiga bagian, yaitu bagian Apache *web* server, pemetaan HTTP-CoAP, dan CoAP client. Sedangkan pada bagian CoAP server, secara garis besar program terdiri dari dua bagian, yaitu CoAP server dan modul komunikasi serial ke mikrokontroler Arduino.

Pada bagian Apache *web* server, agar dapat menerima HTTP *request* dengan bentuk yang distandarkan IETF (tahap *draft*, belum memiliki kode RFC) diperlukan *URL rewrite* yang ditulis dalam file `.htaccess` dalam root apache *web* server folder. Isi dari *URL rewrite* tersebut mengarahkan agar setelah directory `/hc` maka akan dianggap sebagai *query string* dari URL. Isi dari file `.htaccess` tersebut ditunjukkan pada Gambar 4.8.

```

RewriteEngine On
RewriteCond %{THE_REQUEST} \s/+hc/(.*) [NC]
RewriteRule ^ cgi-bin/index.cgi?coap_target_uri=%1
[L,QSA]

```

Gambar 4.8 Konfigurasi *URL Rewrite*

Bagian program pertama adalah program CGI yang diletakkan pada *directory* `/usr/lib/cgi-bin` yang dieksekusi saat ada *HTTP request*. Program ini akan melakukan *request* ke program pemetaan HTTP-CoAP secara *Inter Process Communication* dengan metode UNIX domain *socket*. Setelah menerima data, hasilnya akan diencode menjadi *JSON string* dan akan dienkapsulasi dengan protokol HTTP. Program ini menspesifikasikan tipe konten (`Content-Type: application/json`) dan juga status *response code* HTTP. Tabel 4.1 memberi gambaran mengenai pemetaan respon dari CoAP ke HTTP yang digunakan dalam program ini.

Tabel 4.1 Pemetaan CoAP *Response* ke HTTP *Response*

CoAP response	HTTP response
2.05 Content	200 OK
4.00 Bad Request	400 Bad Request
4.01 Unauthorized	403 Forbidden
4.04 Not Found	404 Not Found
5.04 Gateway Timeout	504 Gateway Timeout

Program CGI diimplementasikan dengan algoritma yang diberikan pada Gambar 4.9.

```

Input: HTTP Query String
Output: Data dalam JSON

TAMBAH interface 6LoWPAN dalam Query String
Socket(AF_UNIX, SOCK_STREAM)
CONNECT(socket)
SEND Query String ke program pemetaan HTTP-CoAP
RECEIVE data
ADD HTTP Header-> Content-Type: application/json
ADD HTTP Header-> Status: X //tergantung CoAP server
ENCODE to JSON
END

```

Gambar 4.9 *Pseudocode* Program CGI

Program pemetaan HTTP-CoAP akan mendengarkan *request* dari program CGI dengan model UNIX domain *socket*. Jika ada permintaan dari CGI, maka satu *request* tersebut akan dilayani dengan satu *thread*. *Thread* tersebut diset agar agar dapat melepaskan diri sendiri dan membuang resource yang digunakan saat *thread* tersebut selesai menyelesaikan *routine* yang diberikan ke *thread*. Di *thread* tersebut juga akan dilakukan pengecekan *cache*, permintaan ke CoAP server, dan penghapusan *cache*. Tiap hal tersebut dikategorikan sebagai *critical section* dan harus dikunci dengan mutex. Gambar 4.10 memberi gambaran mengenai implementasi program dalam bentuk *pseudocode*.

```

Input: UNIX Domain Socket request dari CGI Program
Output: Data dari cache/CoAP server

Socket(AF_UNIX, SOCK_STREAM)
Bind(socket)
Listen(socket)
FOR tiap socket request DO
    ACCEPT connection from CGI
    PTHREAD_CREATE(routine: handle_request)
ENDWHILE

```

Gambar 4.10 *Pseudocode* Program Pemetaan HTTP-CoAP

Masih dalam program pemetaan HTTP-CoAP, terdapat suatu fungsi yang berperan sebagai pemeta HTTP ke CoAP. Fungsi tersebut bernama *handle\_request* dan dijalankan oleh satu *thread*. Fungsi ini membawa nilai

dari `accept()` sebagai parameter agar dapat mengembalikan data ke program CGI yang meminta permintaan HTTP-CoAP. Gambar 4.11 menggambarkan implementasi dari fungsi ini dalam bentuk *pseudocode*.

```

Routine: handle_request

MUTEX_LOCK
CEK_CACHE(par: QUERY STRING)
MUTEX_UNLOCK

IF (ADA DI CACHE)
    KIRIM data ke program CGI
ELSE
    EXEC CoAP_CLIENT(par: QUERY STRING)
    MUTEX_LOCK
    Simpan ke cache
    MUTEX_UNLOCK
    KIRIM data ke program CGI
    Wait(par: Max-Age)
    MUTEX_LOCK
    DELETE data cache
    MUTEX_UNLOCK
ENDIF

```

Gambar 4.11 *Pseudocode* Penanganan Request dalam Satu Thread

*Cache* diimplementasikan sebagai suatu struktur data linked list yang memiliki atribut yang ditunjukkan pada potongan kode pada Gambar 4.12. Atribut `query[256]` menyimpan *query string* dari permintaan ke CoAP server, misalnya `coap://fe80::9090:890/sensor`. Atribut data menyimpan data hasil dari CoAP server, atribut `timeStamp` berisi kapan data *cache* ini dibuat. Jika data *cache* ini sudah habis umurnya, maka akan dihapus dari sistem.



```

struct node {
    char query[256];
    int data;
    unsigned int timeStamp;
    char timeStampStr[32];
    struct node *next;
}

```

Gambar 4.12 Struktur Data *Linked List* untuk *Cache*

Bentuk *cache* yang diajukan dengan struktur *linked list* mengandalkan `malloc()` dan `delete()` untuk membuat dan menghapus data *cache*, oleh karena itu sangat rawan terjadi *memory leak*, suatu kejadian dimana saat program meminta suatu blok memory dengan `malloc()` namun tidak membebaskannya dengan `delete()` saat tidak diperlukan lagi. Data *cache* merupakan data global, yang artinya tiap *thread* bisa saja mengaksesnya dan menyebabkan data tidak konsisten bahkan penghapusan data *node* di *linked list* tersebut. Namun, karena tiap *thread* ingin mengakses data yang ada di *cache*, diperlukan kunci yang ditangani oleh *mutex*. Program ini sudah dites menggunakan program Valgrind untuk mencari *memory leak* dan hasilnya tidak terjadi *memory leak* seperti yang dapat dilihat pada Gambar 4.13.

```

==4202==
==4202== HEAP SUMMARY:
==4202==      in use at exit: 0 bytes in 0 blocks
==4202==    total heap usage: 3 allocs, 3 frees, 912 bytes allocated
==4202==
==4202== All heap blocks were freed -- no leaks are possible
==4202==

```

Gambar 4.13 Hasil Pemeriksaan Program *Cache* dengan Valgrind

Modul CoAP client dibuat dalam bahasa C dan menggunakan *library* libcoap. Tahapannya mirip dengan model internet *socket*, namun payload juga dienkapsulasi dengan *library* libcoap agar dapat berkomunikasi dengan protokol CoAP. Program ini menspesifikasikan *transport layer* yang digunakan, yaitu menggunakan UDP. Setelah itu jika request yang diberikan melalui *network interface* 802.15.4 maka akan payload juga akan dienkapsulasi dengan 6LoWPAN *adaptation layer*.



Selanjutnya adalah bagian pada CoAP server. Pada CoAP server, terdapat dua bagian modul program, yaitu modul CoAP server dan modul komunikasi serial ke Arduino.

Pada modul CoAP server, modul ini akan menunggu hingga ada *request* dari *proxy*. Saat ada *request*, modul akan mencari *resource* yang sudah disediakan oleh server. Modul tersebut adalah sensor dan aktuator yang sudah disediakan dan dapat diakses. Jika tidak ditemukan maka kode 4.04 Not Found akan dikembalikan sebagai respon dari CoAP server. Jika *resource* ditemukan, maka akan dipanggil modul komunikasi serial ke Arduino untuk meminta atau set data. Di modul ini jika dilakukan pemberian atribut *header* dari paket CoAP seperti umur maksimum data untuk kepentingan *caching* (*Max-Age*), kode respon dari CoAP, dan data *resource discovery*. Gambar 4.14 menggambarkan *pseudocode* dari implementasi modul CoAP server.

```

Input: Socket request dari proxy
Output: Data sensor

OPEN Serial ke Arduino
INIT Socket
ADD CoAP attribute
Socket(AF_UNIX, SOCK_STREAM)
Bind(socket)
Listen(socket)
FOR tiap socket request DO
    ACCEPT connection from proxy
    GET data sensor OR SET aktuatoru
    IF (request == sensor)
        SEND data ke proxy
    ENDIF
ENDWHILE

```

Gambar 4.14 *Pseudocode* Program CoAP Server

Modul komunikasi serial ke Arduino terjadi secara dua arah, saat ingin mendapatkan data, Raspberry Pi akan memberi perintah ke Arduino dan Arduino akan menerjemahkan perintah tersebut dan mengembalikan data sensor yang diminta atau mengeset aktuator sesuai perintah. Tiap fitur yang ada pada CoAP server yang direalisasikan melalui Arduino diset dengan *resource*. Terdapat tiga

*resources* pada CoAP server, yaitu nilai sensor ADC untuk keperluan testing dan debugging, sensor *flowmeter* untuk keperluan aplikasi pengukuran debit air pada pipa, dan aktuator berupa *solenoid valve* untuk keperluan aplikasi penutup aliran pipa. Gambar 4.15 menggambarkan aplikasi Arduino yang direpresentasikan dalam *pseudocode*.

```

Input: String dari komunikasi serial
Output: Data sensor atau set nilai ke GPIO

FOR tiap string input serial DO
  IF (Minta data ADC)
    READ ADC
    SEND to Raspberry Pi
  ELSE IF (Minta data flowmeter)
    READ data flowmeter
    SEND to Raspberry Pi
  ELSE IF (Set aktuator)
    SET GPIO(par: 1 atau 0)
  ENDIF
ENDFOR

```

Gambar 4.15 *Pseudocode* Program CoAP Server

## 4.2 Tes Fungsionalitas

Pada bagian ini akan diberikan contoh dari hasil implementasi pemetaan HTTP-CoAP. Pada penelitian ini, pemetaan fokus pada metode GET yang bisa dilakukan semua *platform* diatas arsitektur RESTful. Hal ini memungkinkan sistem untuk berjalan dengan fleksibel jika ingin diintegrasikan menjadi sebuah sistem *Internet of Things*. Terdapat tiga *resources* yang ada di CoAP server dan bisa diakses oleh HTTP *client*.

*Resource* pertama adalah sensor untuk testing, yaitu hasil dari pembacaan ADC mikrokontroler Arduino. Pada realisasinya, nilai analog yang dibaca berasal dari nilai potensiometer, dan nilai tersebut diubah menjadi digital oleh mikrokontroler Arduino. Bentuk URL *request* dan respon yang diberikan dapat dilihat pada Gambar 4.16.



Gambar 4.16 Contoh Implementasi URL untuk Meminta Data Sensor

Pada pengaplikasiannya, digunakan sensor *flowmeter* untuk membaca kecepatan arus air agar bisa didapatkan nilai debit air. *Resource* ini merupakan *resource* kedua yang dimiliki CoAP server. Data dari *resource* ini berupa debit dalam satuan mL/detik.

*Resource* yang ketiga merupakan aktuator yang berupa *solenoid valve*. Karena aktuator diset berdasarkan nilai yang diberikan, maka nilai yang diberikan diletakkan pada *query string* dari CoAP URL, sesuai dengan metode GET. Gambar 4.17 memberikan hasil dari set nilai ke aktuator. Sebuah bilangan 1 atau 0 harus diberikan ke *query string* dari CoAP URL untuk menentukan nilai yang diberikan ke aktuator. Nilai 1 merepresentasikan nilai HIGH, 0 merepresentasikan LOW.

http://alamat-proxy-ipv4/hc/coap://[alamat-CoAP-server-IPv6]/valve?val=nilai

↑  
val=1 atau val=0

Gambar 4.17 Contoh Implementasi GET untuk Set GPIO

*Resource discovery* dapat diminta dengan *path* /.well-known/core. Dengan meminta *resource discovery*, akan didapat data-data *resource* pada CoAP server dalam bentuk JSON. Gambar 4.18 memberikan hasil implementasi *resource discovery*.

Sampler result	Request	Response data
<pre>[{"href":"/","title":"General Info","ct":"0"}, {"href":"/adc","rt":"sensor-adc","title":"Sensor ADC","ct":"50"}, {"href":"/flowmeter","rt":"sensor-flowmeter","title":"Flowmeter Sensor","ct":"50"}, {"href":"/gpio","title":"Set GPIO","ct":"0"}, {"href":"/valve","title":"Set valve","ct":"0"}, {"href":"/pwm","title":"Aktuator PWM","ct":"0"}]</pre>		

Gambar 4.18 Hasil *Resource Discovery* dalam Bentuk JSON

*Proxy* yang diimplementasikan tidak mendukung CoAP *multicast* dan akan memberikan respon 403 Forbidden jika ada *client* yang meminta pemetaan dengan alamat IPv6 *multicast* ke jaringan sensor. Gambar 4.20 merupakan contoh respon yang diberikan jika ada *request* CoAP *multicast* dengan *client* berupa komputer dengan *browser* Mozilla Firefox.

Inspector	Console	Debugger	Style Editor	Performance
✓ Method	File	Domain		
403 GET	/hc/coap://[ff02::]/	192.168.43.51		

Gambar 4.19 Contoh Respon CoAP *Multicast*

Fungsi utama dari *border router* yang berfungsi sebagai HTTP-CoAP *proxy* ini adalah pemetaan HTTP-CoAP. Salah satu komponen dari pemetaan tersebut adalah pemetaan kode respon dari HTTP-CoAP. Tabel 4.2 memberikan status hasil dari pengetesan pemetaan kode respon HTTP-CoAP dan juga ringkasan dari hasil tes fungsional yang dilakukan.

Tabel 4.2 Hasil Tes Fungsional

Fungsi	Status	Jumlah Percobaan
Pemetaan <i>response code</i> CoAP 2.05 ke HTTP 200	berhasil	5
Pemetaan <i>response code</i> CoAP 4.00 ke HTTP 404	berhasil	5
Pemetaan <i>response code</i> CoAP 4.01 ke HTTP 403	berhasil	5
Pemetaan <i>response code</i> CoAP 5.04 ke HTTP 504	berhasil	5
Pegambilan resource sensor data analog (GET)	berhasil	> 100
Pegambilan resource sensor data <i>flowmeter</i> (GET)	berhasil	5
Set resource aktuator melalui GPIO (GET dengan <i>query string</i> )	berhasil	5
<i>caching</i> permintaan GET	berhasil	> 50

### 4.3 Evaluasi Performa

Pada subbab ini, penulis memberikan diskusi dan evaluasi dari performa HTTP-CoAP *proxy* dalam skenario implementasi. Hal ini diperlukan untuk memprofilkan karakteristik dari sistem. Tes yang dilakukan berfokus pada kemampuan HTTP-CoAP *proxy* dalam menghadapi *concurrent requests* yang diukur berdasarkan nilai *latency* dan *throughput* yang dihasilkan saat menghadapi permintaan yang bersamaan dari beberapa *client* sekaligus. Sebagai tambahan, akan juga dilakukan evaluasi terhadap penggunaan memory dari program yang berjalan secara dinamis meminta beberapa blok memori, yaitu program *cache* dan modul pemetaan HTTP-CoAP.

Dalam skenario pengujian, HTTP *client* melakukan permintaan GET ke CoAP server secara *concurrent* menggunakan *tool* Apache JMeter. Permintaan HTTP yang dilakukan HTTP *client* dapat dilihat pada Gambar 4.20.



Gambar 4.20 Contoh Request yang Dilakukan Saat Pengetesan

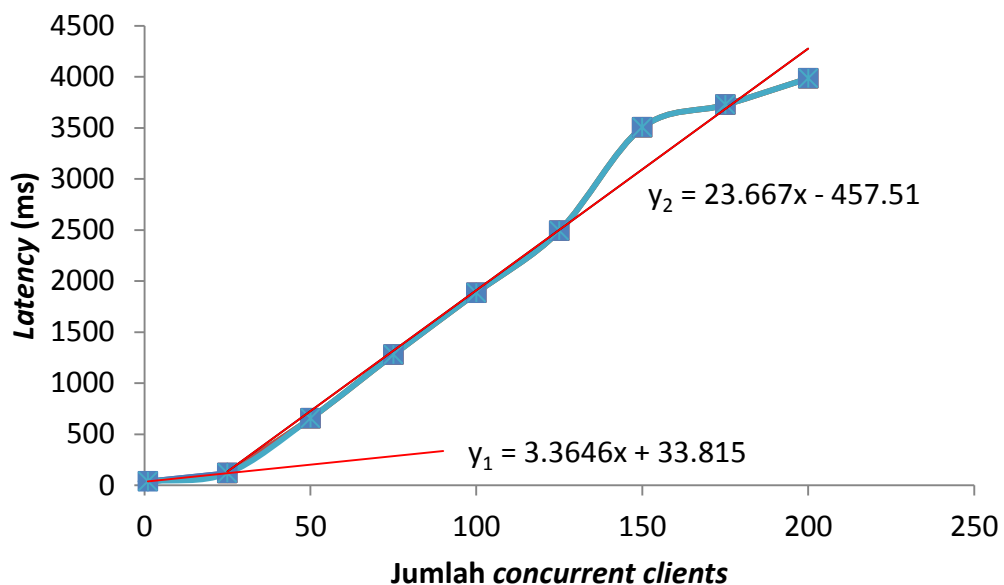
Respon yang diterima HTTP *client* berbentuk JSON *string* seperti yang terlihat pada Gambar 4.21. Dengan Apache JMeter, tidak hanya data respon yang dapat dilihat, namun juga bentuk HTTP *request* dan juga HTTP *header*.

Sampler result	Request	Response data
		{"sensor" : 168}

Gambar 4.21 Data Respon dalam Apache JMeter

#### 4.3.1 Evaluasi *Latency* untuk *Concurrent Client* dengan *Load Testing*

*Latency* merupakan waktu yang dibutuhkan sistem dari mulai dilakukannya permintaan hingga didapatkannya respon dari permintaan tersebut tanpa memperhitungkan waktu yang dibutuhkan untuk mengolah (*render*) respon yang didapat. Pengetesan ini juga memasukkan fitur *caching* di dalam *proxy* dengan umur data 30 detik, sehingga tidak semua *request* diteruskan ke CoAP server. Gambar 4.22 menggambarkan hasil dari pengujian permintaan ke *proxy* secara *concurrent*. Data ini diambil dengan memvariasikan jumlah *client* yang mengakses secara bersamaan dari 1 hingga 200 *clients* dengan selisih kenaikan pengetesan jumlah *clients* sebanyak 25 *clients*.

Gambar 4.22 Jumlah *Clients* vs *Latency*

Pada *chart* yang ditunjukkan Gambar 4.22, peningkatan nilai *latency* terjadi secara *piecewise linear*, dimana peningkatan nilai *latency* pada jumlah *client* antara 1 hingga 25 tidak seragam dengan kenaikan *latency* pada jumlah *client*

diatas 25. Oleh karena itu, garis tren dibuat terpisah. Bagian yang mendekati linear adalah saat diakses oleh lebih dari 25 *concurrent clients*. Tren peningkatan berjalan secara linear seperti yang diberikan pada garis  $y_2$ . Peningkatan yang terjadi naik secara linear dengan koefisien 23.667. Dengan grafik ini, dapat diambil kesimpulan bahwa nilai *latency* akan meningkat secara linear seiring dengan jumlah *client* yang mengakses secara bersamaan.

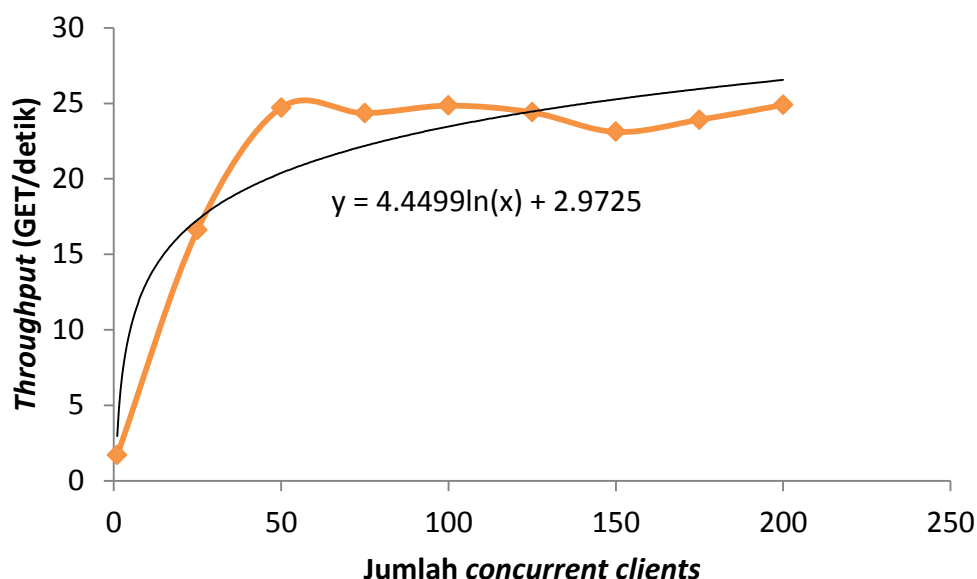
Penanganan tiap *request* dari *client* dilakukan oleh satu *thread* dan untuk mengakses CoAP server hanya diperbolehkan satu *request* ke *constrained network*. Data kenaikan *latency* ini menunjukkan bahwa nilai *latency* dipengaruhi oleh jumlah *client* yang melakukan *request* karena *requests* tersebut akan bergantian melakukan permintaan ke CoAP server dan/atau mengakses *cache* dengan menunggu kunci yang dilepas oleh *mutex*.

#### 4.3.2 Evaluasi *Throughput* Dengan *Load Testing*

Dalam konteks ini, *throughput* dihitung dengan jumlah permintaan/satuan waktu. Waktu yang dihitung merupakan waktu dari saat *client* pertama melakukan permintaan hingga *client* terakhir mendapatkan respon dari permintaannya. Persamaan 4.1 memberikan persamaan untuk mencari *throughput* [33].

$$throughput = \frac{jumlah\ permintaan}{waktu\ total} \quad (4.1)$$

Gambar 4.23 menunjukkan nilai *throughput* sebagai fungsi dari jumlah *clients* yang mengakses HTTP-CoAP *proxy*. Nilai *throughput* meningkat secara logaritmis seiring jumlah *client* yang mengakses, hingga jumlah *clients* yang mengakses sebanyak 50. Diatas 50 *clients*, nilai *throughput* berlangsung relatif stabil, dengan nilai fluktuasi yang tidak terlalu signifikan.



Gambar 4.23 Jumlah *Clients* vs *Throughput*

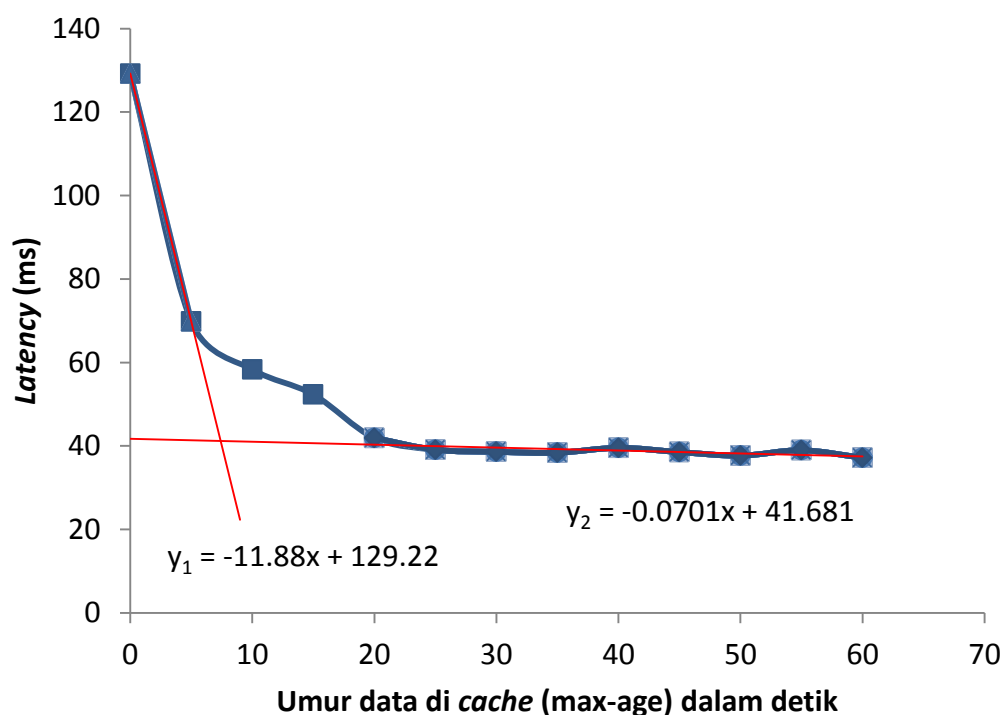
Gambar 4.23 juga menunjukkan bahwa nilai *throughput* mendekati konstan jika di atas 50 *clients*. *Throughput* pada konteks ini dapat menunjukkan berapa *requests* yang dapat ditangani oleh *proxy* per detik. Dengan mencari rata-rata yang dibulatkan ke bawah dari *throughput* untuk setiap titik pengujian, didapatkan nilai 23 *requests* per detik (atau 1380 *requests* per menit) yang dapat ditangani oleh HTTP-CoAP *proxy* yang dirancang dan diimplementasikan.

#### 4.3.3 Evaluasi Penggunaan *Cache*

Metrik yang menjadi parameter dalam pengukuran performa *cache* adalah *hit rate*, *latency*, dan *bandwidth saving* [34]. Dalam *cache* yang dirancang, mengukur *hit rate* tidaklah relevan untuk mengukur performa *cache*. Hal tersebut dikarenakan nilai *hit rate* yang bergantung pada umur maksimal dari data pada *cache*, bukan dari metode *replacement* pada *cache*. Sebagai gantinya, mengukur *latency* dapat menjadi acuan dari pengaruh penggunaan *cache* dalam *proxy*. Kelebihan dari mengukur *latency* sebagai salah satu pengukuran efektivitas *cache* sebagai mekanisme *congestion control* adalah sisi pengguna dapat merasakan dampak langsung dari penggunaan *proxy*, karena kenyatannya pengguna lebih peduli terhadap waktu eksekusi dibandingkan dengan *hit rate* dan *bandwidth saving*.



Dalam pengujiannya, satu HTTP *client* mengakses CoAP server sebanyak 100 kali melalui *proxy* selama 60 detik (jeda antar *request* 1,67 detik). Gambar 4.24 menggambarkan pengaruh umur data di *cache* dengan *latency* yang didapat *client*. Nilai *latency* maksimum diberikan jika umur data di *cache* selama 0 detik, artinya tidak menggunakan *cache* (*hit rate*: 0%). Jika umur data dipanjangkan (*hit rate* meningkat), maka *latency* akan turun, yang mengindikasikan bahwa *traffic* ke *constrained network* juga berkurang. Namun, penurunan menjadi sangat lambat saat umur data *cache* diatas 20 detik, dengan skala penurunan -0.0701 (mendekati 0). Dapat diasumsikan bahwa diatas umur *cache* 20 detik nilai *latency* relatif stabil.



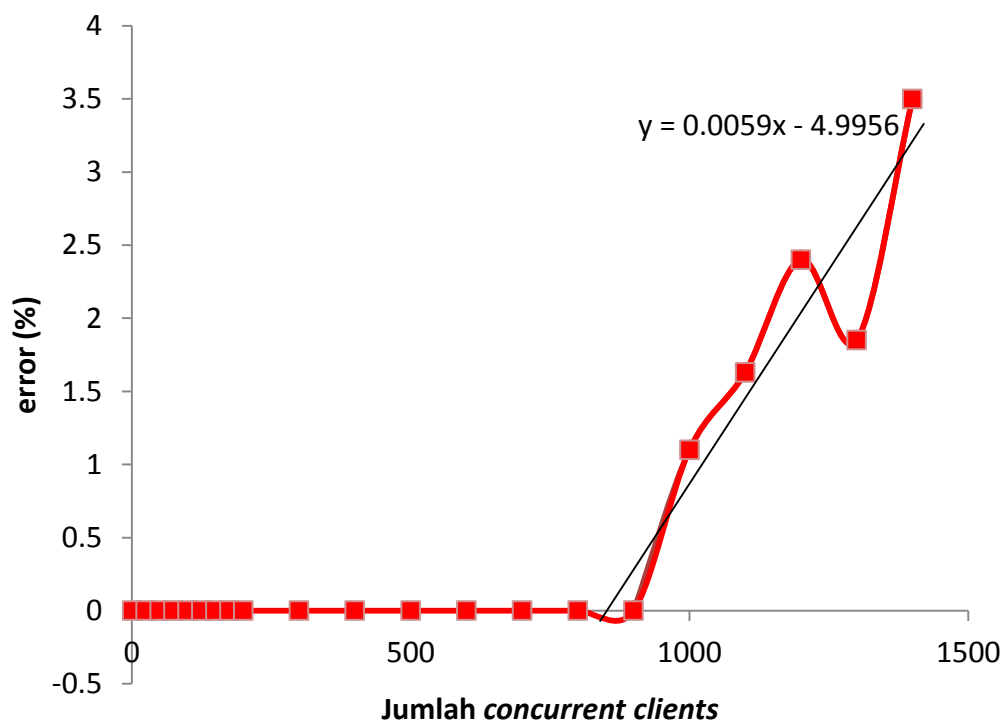
Gambar 4.24 Grafik Umur Data *Cache* vs *Latency*

Grafik pada Gambar 4.24 dapat mengindikasikan ada dua garis tren yang dapat diambil, direpresentasikan dengan garis linear  $y_1$  dan  $y_2$ . Garis  $y_1$  menunjukkan tren penurunan nilai *latency*, garis  $y_2$  menunjukkan tren yang stabil dari nilai *latency*. Walaupun garis  $y_2$  menunjukkan nilai *latency* pada titik terendahnya, hal ini tidak selamanya menguntungkan karena memungkinkan data dari CoAP server tidak *fresh* atau valid lagi. Oleh karena itu, dapat disimpulkan,

nilai yang optimal untuk umur *cache* adalah diantara 5 hingga 20 detik, seperti yang diapit oleh dua garis tren pada grafik di Gambar 4.24.

#### 4.3.4 Evaluasi *Error Rate* yang Dihasilkan dengan *Stress Testing*

Nilai *error rate* penting untuk dianalisa untuk memprofilkan karakteristik dari HTTP-CoAP *proxy*. Pengetesan dilakukan dengan membebani *proxy* dengan meningkatkan jumlah *clients* yang meminta secara bersamaan dan dilihat apakah ada permintaan yang gagal dilayani. Gambar 4.25 menunjukkan *error rate* sebagai fungsi jumlah *clients*.



Gambar 4.25 Jumlah *Clients* vs *Error*

Pada Gambar 4.24, *error* tidak terjadi saat jumlah permintaan dari *clients* dibawah 1000. Akan tetapi, jika permintaan sudah sampai 1000 keatas, terjadi *error* dimana tidak berhasilnya permintaan *client* yang dilayani. Kenaikan nilai *error* terus terjadi secara tajam seiring naiknya jumlah *clients* yang mengakses diatas 1000 *client* dengan koefisien kemiringan mendekati 0 (0.0059). Jadi, dapat dikatakan bahwa tidaklah aman jika memberi lebih dari 1000 *request* ke *proxy* dalam waktu bersamaan.

Error yang terjadi ditunjukkan pada Gambar 4.26. Pada gambar tersebut, urutan koneksi TCP yang terjadi antara HTTP *client* dengan *proxy*.

No.	Time	Source	Destination	Prot	Len	Info
4725	16.2137	10.42.0.1	10.42.0.22	TCP	74	42784 > http [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=14555179 T
7166	17.2120	10.42.0.1	10.42.0.22	TCP	74	[TCP Retransmission] 42784 > http [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK PE
15805	19.2159	10.42.0.1	10.42.0.22	TCP	74	[TCP Retransmission] 42784 > http [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK PE
15846	19.2187	10.42.0.22	10.42.0.1	TCP	74	http > 42784 [SYN, ACK] Seq=0 Ack=1 Win=28960 Len=0 MSS=1460 SACK_PERM=1 TSval
15841	19.2187	10.42.0.1	10.42.0.22	TCP	66	42784 > http [ACK] Seq=1 Ack=1 Win=29312 Len=0 TSval=14555930 TSecr=99415
15842	19.2188	10.42.0.1	10.42.0.22	HTTP	208	GET /hc/coap://%5Bfe80::1:2%5D/adc HTTP/1.1
17289	19.4200	10.42.0.1	10.42.0.22	HTTP	208	[TCP Retransmission] GET /hc/coap://%5Bfe80::1:2%5D/adc HTTP/1.1
18676	19.6320	10.42.0.1	10.42.0.22	HTTP	208	[TCP Retransmission] GET /hc/coap://%5Bfe80::1:2%5D/adc HTTP/1.1
20597	20.0560	10.42.0.1	10.42.0.22	HTTP	208	[TCP Retransmission] GET /hc/coap://%5Bfe80::1:2%5D/adc HTTP/1.1
22257	20.9040	10.42.0.1	10.42.0.22	HTTP	208	[TCP Retransmission] GET /hc/coap://%5Bfe80::1:2%5D/adc HTTP/1.1
25772	22.6040	10.42.0.1	10.42.0.22	HTTP	208	[TCP Retransmission] GET /hc/coap://%5Bfe80::1:2%5D/adc HTTP/1.1
29062	26.0080	10.42.0.1	10.42.0.22	HTTP	208	[TCP Retransmission] GET /hc/coap://%5Bfe80::1:2%5D/adc HTTP/1.1
33032	32.8080	10.42.0.1	10.42.0.22	HTTP	208	[TCP Retransmission] GET /hc/coap://%5Bfe80::1:2%5D/adc HTTP/1.1
38447	46.4080	10.42.0.1	10.42.0.22	HTTP	208	[TCP Retransmission] GET /hc/coap://%5Bfe80::1:2%5D/adc HTTP/1.1
43141	73.6400	10.42.0.1	10.42.0.22	HTTP	208	[TCP Retransmission] GET /hc/coap://%5Bfe80::1:2%5D/adc HTTP/1.1
47631	128.040	10.42.0.1	10.42.0.22	HTTP	208	[TCP Retransmission] GET /hc/coap://%5Bfe80::1:2%5D/adc HTTP/1.1
47718	128.044	10.42.0.22	10.42.0.1	TCP	60	http > 42784 [RST] Seq=1 Win=0 Len=0

Gambar 4.26 Hasil Request yang Bersifat Error Pada Wireshark

Gambar 4.26 menunjukkan sebuah TCP *connection* yang dilakukan saat melakukan *request*. Hal yang membuat *request* gagal adalah *connection reset* yang ditunjukkan pada baris terakhir dari Gambar 4.26. *Connection reset* dilakukan pada TCP agar *resource* yang ada tidak terlalu lama ditahan sehingga *request* berikutnya bisa dilayani. Sebelum terjadi *connection reset*, terjadi TCP *Retransmission* sebanyak 10 kali. Banyaknya TCP *Retransmission* yang dilakukan bergantung pada TCP *keepalive probes*, yang secara *default* diatur sebanyak 10 kali. Hal yang perlu diperhatikan adalah terjadinya TCP *Retransmission* yang mengindikasikan kegagalan koneksi pada percobaan pertama. Faktor utama penyebab TCP *Retransmission* adalah *network congestion*. Dari analisa error ini, dapat disimpulkan jika sudah diatas 1000 *clients* yang meminta bersamaan ke *proxy*, terjadi *network congestion* sehingga *request* dilayani dalam waktu yang lama, bahkan butuh TCP *Retransmission*. Kegagalan yang terjadi dikarenakan dibatasinya TCP *Retransmission* yang berguna untuk menjaga agar *proxy* tetap bisa melayani *request* yang lain.

Hasil dari evaluasi performa dapat dijadikan acuan untuk membangun spesifikasi sistem *Internet of Things*. Dalam membangun spesifikasi sistem *Internet of Things*, beberapa hal yang perlu diperhatikan adalah *latency* untuk menentukan sifat *real-time* dari sistem, *throughput* untuk menentukan jumlah

*concurrent clients* yang dapat ditangani sistem, dan maksimum jumlah *concurrent clients* yang boleh mengakses sistem. Dari analisis performa dari proxy yang dirancang, proxy dapat memuaskan 23 permintaan bersamaan dari *client* yang berberda per detik, dengan kenaikan *latency* sebesar 23.667, dan tidak bisa bekerja optimal jika diakses lebih dari 1000 *client* secara bersamaan. Dari karakter *proxy* yang didapat dari analisa tersebut, dapat dikatakan proxy cocok untuk aplikasi yang memiliki pengguna yang tidak terlalu banyak dan bersifat *soft deadline*, seperti *smart home*, aplikasi pemantauan pada rumah, pipa air, tanaman, kondisi bangunan, dan sebagainya.

## BAB 5

### KESIMPULAN DAN SARAN

Berdasarkan hasil implementasi yang sudah dibahas pada bab-bab sebelumnya, terdapat beberapa kesimpulan yang dapat diambil:

- Perancangan *border router* untuk jaringan sensor nirkabel berhasil diimplementasikan dengan IEEE 802.15.4 pada lapisan fisik, 6LoWPAN *adaptation layer*, dan HTTP sekaligus CoAP pada lapisan aplikasi sesuai dengan petunjuk dari *draft* IETF untuk pemetaan protokol HTTP ke CoAP [14] dan juga berhasil memformat data respon sesuai dengan *draft* IETF mengenai format data respon dalam JSON [31].
- Hasil dari *load testing* menunjukkan bahwa *proxy* dapat diakses oleh beberapa *clients* secara bersamaan dengan pola kenaikan nilai *latency* yang linear terhadap kenaikan jumlah *clients* yang mengakses secara bersamaan dengan peningkatan 23.667 ms per penambahan *client* yang mengakses.
- Nilai *throughput* naik seiring dengan jumlah *client* yang meningkat namun mengalami saturasi dan cenderung konstan setelah titik saturasi. Nilai *throughput* yang diperoleh menunjukkan bahwa *proxy* dapat menangani 23 permintaan GET per detik.
- Penggunaan *cache* efektif dalam mereduksi nilai *latency*, namun dengan kekurangan tidak validnya data jika terlalu lama disimpan di *cache*. Hasil analisa grafik menunjukkan umur data 5 hingga 20 detik merupakan nilai umur *cache* yang optimal untuk diterapkan.
- *Stress Testing* dilakukan untuk menguji ketahanan yang sesungguhnya dari *proxy* dengan parameter jumlah error yang muncul jika diakses banyak *client* secara bersamaan. Error mulai terlihat saat 1000 *clients* ke atas mengakses *proxy* bersamaan dengan kenaikan yang sangat tajam

dengan peningkatan sebesar 0.0059 per penambahan jumlah *client* yang mengakses. Hal ini menunjukkan tidaklah aman untuk membebani *proxy* dengan lebih dari 1000 *concurrent clients*.

- Hasil analisa performa menunjukkan bahwa *proxy* yang dirancang cocok untuk aplikasi yang memiliki pengguna yang tidak terlalu banyak dan bersifat *soft deadline* seperti *smart home* dan berbagai aplikasi pemantauan tanaman, pipa air, dan sebagainya.

Untuk pekerjaan selanjutnya dari penelitian serupa, area keamanan menjadi hal penting untuk dibahas. Oleh karena itu, sebagai saran untuk penelitian selanjutnya terdapat beberapa hal yang dapat dikembangkan lebih jauh seperti:

- Perancangan *proxy* dengan fitur HTTPS dan juga CoAPS dapat menjadi salah satu fitur penting agar komunikasi dapat terenkripsi.
- *Proxy* juga bisa diberikan mekanisme untuk mendeteksi *traffic overload* agar mencegah serangan *Denial-of-Service*.
- Otentikasi *client* yang mengakses *constrained network* sehingga tidak mudah untuk diakses oleh pihak yang tidak memiliki wewenang.

## DAFTAR REFERENSI

- [1] J. Höller, Ed., *From machine-to-machine to the Internet of things: introduction to a new age of intelligence*. Amsterdam: Elsevier Academic Press, 2014.
- [2] A. Whitmore, A. Agarwal, dan L. D. Xu, “The Internet of Things—A survey of topics and trends,” *Inf. Syst. Front.*, vol. 17, no. 2, hal. 261–274, Mar 2014.
- [3] Z. Shelby dan C. Bormann, *6LoWPAN: The Wireless Embedded Internet*. Wiley, 2009.
- [4] G. Gardasevic, S. Mijovic, A. Stajkic, dan C. Buratti, “On the performance of 6LoWPAN through experimentation,” in *2015 International Wireless Communications and Mobile Computing Conference (IWCMC)*, 2015, hal. 696–701.
- [5] B. da S. Campos, J. J. P. C. Rodrigues, L. D. P. Mendes, E. F. Nakamura, dan C. M. S. Figueiredo, “Design and Construction of Wireless Sensor Network Gateway with IPv4/IPv6 Support,” in *2011 IEEE International Conference on Communications (ICC)*, 2011, hal. 1–5.
- [6] J. T. Adams, “An introduction to IEEE STD 802.15.4,” in *2006 IEEE Aerospace Conference*, 2006, hal. 8 pp.-pp.
- [7] A. Koubâa, M. Alves, dan E. Tovar, “IEEE 802.15.4 for Wireless Sensor Networks: A Technical Overview,” CISTER - Research Centre in Realtime and Embedded Computing Systems, Jul 2005.
- [8] L. F. Schrickte, C. Montez, R. d Oliveira, dan A. R. Pinto, “Integration of Wireless Sensor Networks to the Internet of Things Using a 6LoWPAN Gateway,” in *2013 III Brazilian Symposium on Computing Systems Engineering*, 2013, hal. 119–124.
- [9] J. W. Hui, D. E. Culler, dan S. Chakrabarti, “6LoWPAN: Incorporating IEEE 802.15.4 into the IP architecture,” 2009, vol. 3.

- [10] O. Bergmann, K. T. Hillmann, dan S. Gerdes, “A CoAP-gateway for smart homes,” in *2012 International Conference on Computing, Networking and Communications (ICNC)*, 2012, hal. 446–450.
- [11] C. Lerche, N. Laum, F. Golasowski, D. Timmermann, dan C. Niedermeier, “Connecting the web with the web of things: lessons learned from implementing a CoAP-HTTP proxy,” in *2012 IEEE 9th International Conference on Mobile Adhoc and Sensor Systems (MASS)*, 2012, vol. Supplement, hal. 1–8.
- [12] Z. Shelby, “Embedded web services,” *IEEE Wirel. Commun.*, vol. 17, no. 6, hal. 52–57, Des 2010.
- [13] Z. Shelby, K. Hartke, dan C. Bormann, “The Constrained Application Protocol (CoAP),” Jun-2014. [Daring]. Tersedia pada: <https://tools.ietf.org/html/rfc7252>. [Diakses: 24-Nov-2015].
- [14] A. Castellani, S. Loreto, A. Rahman, T. Fossati, dan E. Dijk, “Guidelines for HTTP-to-CoAP Mapping Implementations draft-ietf-core-http-mapping-10.” [Daring]. Tersedia pada: <https://tools.ietf.org/html/draft-ietf-core-http-mapping-10>. [Diakses: 04-Jun-2016].
- [15] R. E. Bryant dan D. R. O’Hallaron, *Computer Systems: A Programmer’s Perspective*, 2 edition. Boston: Pearson, 2010.
- [16] B. Hall, *Beej’s Guide to Network Programming*. Jorgensen Publishing.
- [17] R. Rosen, *Linux Kernel Networking: Implementation and Theory*, 2014 edition. New York, NY: Apress, 2013.
- [18] D. R. Butenhof, *Programming with POSIX Threads*. Reading, Mass: Addison-Wesley Professional, 1997.
- [19] D. Robbins, “Common threads: POSIX threads explained, Part 2,” 01-Agu-2000. [Daring]. Tersedia pada: <http://www.ibm.com/developerworks/library/l-posix2/>. [Diakses: 21-Mei-2016].



- [20] “RPi Hardware - eLinux.org.” [Daring]. Tersedia pada: [http://elinux.org/RPi\\_Hardware](http://elinux.org/RPi_Hardware). [Diakses: 04-Jun-2016].
- [21] C. P. Kruger dan G. P. Hancke, “Benchmarking Internet of things devices,” in *2014 12th IEEE International Conference on Industrial Informatics (INDIN)*, 2014, hal. 611–616.
- [22] “MRF24J40MA - Wireless Modules.” [Daring]. Tersedia pada: <http://www.microchip.com/wwwproducts/Devices.aspx?product=MRF24J40MA>. [Diakses: 09-Des-2015].
- [23] Microchip, *MRF24J40MA Data Sheet*. 2008.
- [24] “HTTPD - Apache2 Web Server.” [Daring]. Tersedia pada: <https://help.ubuntu.com/lts/serverguide/httpd.html>. [Diakses: 04-Jun-2016].
- [25] “The Apache HTTP Server Open Source Project on Open Hub: Languages Page.” [Daring]. Tersedia pada: [https://www.openhub.net/p/apache/analyses/latest/languages\\_summary](https://www.openhub.net/p/apache/analyses/latest/languages_summary). [Diakses: 04-Jun-2016].
- [26] J. Hamilton, *CGI Programming 101: Programming Perl for the World Wide Web, Second Edition*, 2 edition. Houston, TX: CGI101.com, 2004.
- [27] N. DuPaul, “Software Development Lifecycle (SDLC),” *Veracode*. [Daring]. Tersedia pada: <http://www.veracode.co.uk/security/software-development-lifecycle>. [Diakses: 25-Nov-2015].
- [28] A. Castellani, T. Fossati, dan S. Loreto, “HTTP-CoAP cross protocol proxy: an implementation viewpoint,” in *2012 IEEE 9th International Conference on Mobile Adhoc and Sensor Systems (MASS)*, 2012, vol. Supplement, hal. 1–6.
- [29] J. Jeong, J. Lee, H. Kim, G. Shin, dan S. Kim, “Zero Configuration HTTP-CoAP Proxy Implementation based on CGI,” in *Pervasive 2012*, Newcastle, UK.

- [30] A. Ludovici dan A. Calveras, “A Proxy Design to Leverage the Interconnection of CoAP Wireless Sensor Networks with Web Applications,” *Sensors*, vol. 15, no. 1, hal. 1217–1244, Jan 2015.
- [31] K. Li, A. Rahman, dan C. Bormann, “Representing CoRE Formats in JSON and CBOR draft-ietf-core-links-json-05.” [Daring]. Tersedia pada: <https://tools.ietf.org/html/draft-ietf-core-links-json-05>. [Diakses: 04-Jun-2016].
- [32] M. Kovatsch, “CoAP for the Web of Things: From Tiny Resource-constrained Devices to the Web Browser,” in *Proceedings of the 2013 ACM Conference on Pervasive and Ubiquitous Computing Adjunct Publication*, New York, NY, USA, 2013, hal. 1495–1504.
- [33] “Apache JMeter - User’s Manual: Glossary.” [Daring]. Tersedia pada: <http://jmeter.apache.org/usermanual/glossary.html>. [Diakses: 04-Jun-2016].
- [34] W. Chan Kit dan M. Somasundaram, “Concept of Cache in Web Proxies.” [Daring]. Tersedia pada: <http://www.webabode.com/articles/Web%20Caching.pdf>. [Diakses: 14-Jun-2016].

## LAMPIRAN

### LAMPIRAN 1

#### HTTP-CoAP Proxy Web Test Bed

## HTTP-CoAP Proxy Test Bed

Proxy's IPv4 Address: 192.168.43.51

CoAP Servers' IPv6 Addresses	ADC	Flowmeter	Aktuator	
fe80::1:2	{"adc" : 628}	{"flowmeter" : 0}	<input type="button" value="Close"/>	<input type="button" value="Open"/>
fe80::1:3	{"adc" : 914}	{"flowmeter" : 0}	<input type="button" value="Close"/>	<input type="button" value="Open"/>

#### CoAP Resource Discovery

CoAP Servers' IPv6 Addresses	Raw	Decoded
<input type="button" value="fe80::1:2"/>	[{"href":"/adc","rt":"sensor-adc","title":"Sensor ADC","ct":"50"}, {"href":"/flowmeter","rt":"sensor-flowmeter","title":"Flowmeter Sensor","ct":"50"}, {"href":"/valve","rt":"aktuator-valve","title":"Set valve","ct":"0"}]	<b>href:</b> /adc <b>rt:</b> sensor-adc <b>title:</b> Sensor ADC <b>ct:</b> 50  <b>href:</b> /flowmeter <b>rt:</b> sensor-flowmeter <b>title:</b> Flowmeter Sensor <b>ct:</b> 50  <b>href:</b> /valve <b>rt:</b> aktuator-valve <b>title:</b> Set valve <b>ct:</b> 0
<input type="button" value="fe80::1:3"/>	[{"href":"/adc","rt":"sensor-adc","title":"Sensor ADC","ct":"50"}, {"href":"/flowmeter","rt":"sensor-flowmeter","title":"Flowmeter Sensor","ct":"50"}, {"href":"/valve","rt":"aktuator-valve","title":"Set valve","ct":"0"}]	<b>href:</b> /adc <b>rt:</b> sensor-adc <b>title:</b> Sensor ADC <b>ct:</b> 50  <b>href:</b> /flowmeter <b>rt:</b> sensor-flowmeter <b>title:</b> Flowmeter Sensor <b>ct:</b> 50  <b>href:</b> /valve <b>rt:</b> aktuator-valve <b>title:</b> Set valve <b>ct:</b> 0